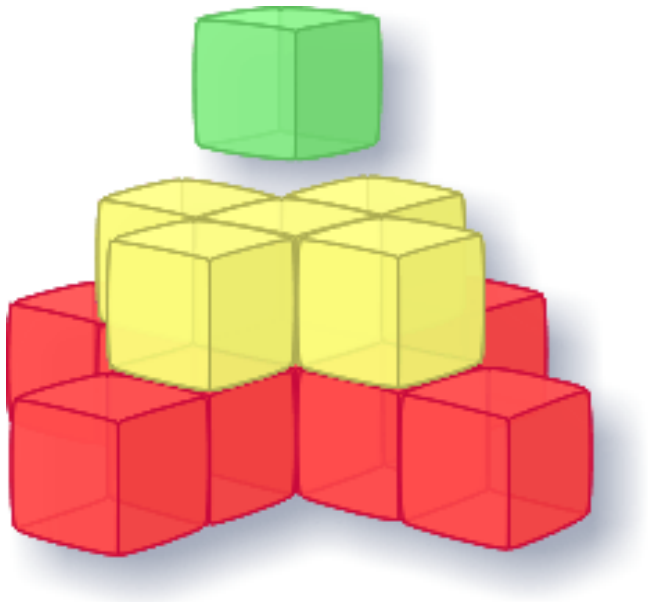
Dipl. Ing. Toralf Riedel
Dipl. Ing. Päd. Alexander Huwaldt

Benutzerhandbuch

SiSy® AVR Mikrocontroller

gültig ab SiSy-Version 2.17



Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Die Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind die Autoren dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Dokument erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden.

3. Auflage: Juni 2006

© Laser & Co. Solutions GmbH
www.laser-co.de
www.myavr.de
info@myavr.de.de
Tel: ++49 (0) 3585 470 222
Fax: ++49 (0) 3585 470 233

Inhalt

1 Einleitung 5
2 Vorbereitung 6
2.1 Beschaffung und Installation der Software 6
2.1.1 Voraussetzungen 6
2.1.2 Setup von der SiSy-CD 6
2.2 Beschaffen bzw. Herstellen der Hardware 8
3 Arbeiten mit SiSy AVR, allgemein 9
3.1 Projektarbeit 9
3.1.1 Was ist ein SiSy-Projekt? 9
3.1.2 Neues Projekt erstellen 9
3.1.3 Vorhandenes Projekt öffnen 9
3.1.4 Projekt archivieren 9
3.1.5 Projektarchiv einlesen 10
3.2 Die Modellierungselemente von SiSy 10
3.3 Die Fenster für die Modellierung 11
3.4 Druckfunktionen in SiSy 12
3.4.1 Diagramme drucken 12
3.4.2 Grafiken und Inhalte drucken (QuickDok) 14
3.4.3 Nur Quellcodes drucken 15
3.4.4 Nutzen der Zwischenablage 16
4 Die Hilfefunktionen in SiSy 17
4.1 Der Assistent 17
4.2 Die Online-Hilfe 18
4.3 Die allgemeine Hilfe 19
5 Entwicklung eines kleinen Programms mit SiSy AVR 20
5.1 Voraussetzungen 20
5.2 Zielstellung 20
5.3 Vorgehen 20
6 Das myAVR Controlcenter 28
6.1 Einleitung 28
6.2 Das myAVR Board starten und stoppen (Power Control) 28
6.3 Kommunikation mit dem myAVR Board 29
6.3.1 Grundlagen (LPT/USB-Variante) 29
6.3.2 Einstellungen für die seriellen Verbindung 30
6.3.3 Daten empfangen vom myAVR Borad 31
6.3.4 Darstellung der empfangen Daten 31
6.3.5 Empfangene Daten speichern 34
6.3.6 Daten an das myAVR Board senden 35
7 Der myAVR Code-Wizard 36
7.1 Einführung 36
7.2 Grundeinstellungen 37
7.3 Geräteeinstellungen 37
7.4 Unterprogramme 38
7.5 Projektdaten 38
7.6 Codegenerierung 39
8 Entwicklung eines großen Programms mit SiSy AVR 40
8.1 Einleitung 40
8.2 Vorbereitung 40

8.3 Aufgabenstellung ... 41
8.4 Hauptprogramm erstellen ... 41
8.5 Units (Unterprogramme) anlegen und verknüpfen ... 42
8.6 Übersetzen, Brennen und Test ... 43
8.7 Interrupt-Service-Routine (ISR) im großen Programm ... 43
9 Entwicklung eines Programmablaufplans mit SiSy AVR ... 44
9.1 Einleitung ... 44
9.2 Vorbereitung ... 44
9.3 Aufgabenstellung ... 46
9.4 Grundstruktur laden ... 46
9.5 Logik entwerfen ... 47
9.6 Befehle eingeben ... 48
9.7 Übersetzen, brennen und Test ... 50
9.8 Unterprogrammtechnik im PAP ... 52
9.8.1 Anlegen eines Unterprogramms ... 52
9.8.2 Ein Unterprogramm aufrufen ... 54
9.8.3 Unterprogramme mehrmals benutzen ... 55
9.9 Interrupt-Service-Routinen (ISR) im PAP ... 56
9.10 Daten im PAP ... 57
9.10.1 Anlegen eines Datenobjektes ... 57
9.10.2 Datenobjekt benutzen ... 57
10 Entwicklung eines Struktogramms mit SiSy AVR ... 58
10.1 Einleitung ... 58
10.2 Aufgabenstellung ... 58
10.3 Vorbereitung ... 59
10.4 Struktogramm entwickeln ... 60
10.5 Übersetzen, brennen und Testen ... 63
10.6 Funktionen (Unterprogramme) im Struktogramm ... 64
10.7 Interrupt-Service-Routinen (ISR) im Struktogramm ... 65
11 Entwicklung eines Klassendiagramms mit SiSy AVR ... 66
11.1 Einleitung ... 66
11.2 Aufgabenstellung ... 67
11.3 Vorbereitung ... 67
11.4 Grundstruktur laden ... 68
11.5 Systemstruktur entwerfen ... 69
11.6 Systemverhalten programmieren ... 72
11.7 Übersetzen, Brennen und Testen ... 73
11.8 Interrupt-Service-Routinen (ISR) im Klassendiagramm ... 74
12 Einstellungen Fuse- und Lock-Bits mit SiSy ... 76
12.1 Einleitung ... 76
12.2 Fuse- und Lockbits, Benutzeroberfläche in SiSy AVR ... 76
12.3 Fuse- und Lockbits verändern ... 78
Anhang: Tastaturbelegung, allgemein ... 80
Anhang: Tastenbelegung im Struktogramm ... 82
Anhang: Mausoperationen ... 83

1 Einleitung

Sie haben eine Ausgabe des Modellierungswerkzeuges Simple System, kurz SiSy, erworben. Bevor auf die verschiedenen Funktionen des Programms eingegangen wird, noch einige Worte zum vorliegenden Handbuch. Mit Hilfe des Handbuchs werden dem Nutzer die Grundlagen der Bedienung von SiSy erläutert. Der Inhalt, die Gestalt und die Regeln der Modelle werden nur am Rand betrachtet. Das genaue Vorgehen für die Programmierung eines Mikroprozessors wird an einem Beispiel ausführlich beschrieben. Auf die Grundlagen der Mikroprozessorprogrammierung wird im Rahmen dieses Handbuches nicht eingegangen. Dazu dienen die myAVR Lehrbücher.
Dem Nutzer wird in diesem Handbuch der Einstieg in das Programm erleichtert und die umfangreichen Funktionen von SiSy kurz und verständlich beschrieben. Bei der Arbeit mit SiSy erstellt der Anwender Modelle in Form von Diagrammen und in ihnen enthaltene Symbole. Die Grundlagen der Entstehung und Bearbeitung solcher Diagramme sind Gegenstand der Betrachtung dieses Handbuchs.
Folgende Darstellungs- und Gestaltungsmittel sind für den Nutzer bei der Arbeit mit SiSy besonders wichtig:

- die Diagramme als Fenster zur Ansicht und Bearbeitung von Modellen;
- der Navigator als Fenster zur Steuerung und Bewegung in Modellen;
- der Assistent mit Hilfestellungen zum jeweils geöffneten Diagramm und mit Diagrammvorlagen (wenn vorhanden);
- die Menüs und Schalter für Befehle an Navigator, Diagramm und Objekt im Kontext mit der Modellierung.

Zu den Bezeichnungen im Text:

- Falls ein Menübefehl nur über Untermenüs zu erreichen ist, werden die einzelnen Menübezeichnungen kursiv geschrieben und durch Schrägstriche voneinander getrennt.
  Beispiel: Menü *Hilfe/über SiSy*

- Titel von Dialogboxen, Schaltflächen und Menüpunkten werden in Anführungszeichen gesetzt.
  Beispiel: Dialogbox „Definition“, Schaltfläche „OK“

2 Vorbereitung

In diesem Kapitel werden Sie über notwendige Schritte zur Beschaffung, Installation, Konfiguration und Aufbau einer funktionsfähigen Entwicklungsumgebung informiert.

2.1 Beschaffung und Installation der Software

Für die Bearbeitung der Übungen und Aufgaben steht Ihnen die AVR-Entwicklungsumgebung SiSy AVR zur Verfügung. Sollten Sie SiSy AVR bereits installiert haben, können Sie dieses Kapitel überspringen.

2.1.1 Voraussetzungen

Für die Installation benötigen Sie einen Freischaltcode (Lizenzangaben). Falls Sie diese Angaben nicht mit der Software erhalten haben, können Sie diese online abrufen von

www.myAVR.de à Service

oder fordern Sie diese beim Hersteller an:

Tel: 03585-470222
Fax: 03585-470233
e-Mail: hotline@myAVR.de.

Außerdem sollten Sie prüfen, ob die Systemvoraussetzungen für die Installation und die Arbeit mit SiSy AVR gewährleistet sind.

- für das Board 1.x einen PC-Arbeitsplatz oder Notebook mit LPT-Port und mindestens einem COM-Port
- für das Board 2.x einen PC-Arbeitsplatz oder Notebook mit USB-Anschluss
- Prozessor ab 800 MHz
- 330 MB freier Speicherplatz auf der Festplatte
- Windows 2000, XP oder 2003
- mindestens 64 MB RAM, empfohlen 256 MB RAM
- Microsoft Internet-Explorer ab Version 6.0
- Maus oder ähnliches Zeigegerät
- Assembler Entwicklungsumgebung (in SiSy bereits integriert)
- myAVR Board
- LPT-Verlängerung (Board 1.x) bzw. USB Kabel (Board 2.x)
- Null-Modemkabel (bei Board 1.x)
- Bei Bedarf (z.B. autonomer Einsatz des myAVR Boards) geeignete Spannungsversorgung z.B. 9 V Batterie/stabilisiertes 9 V Netzteil

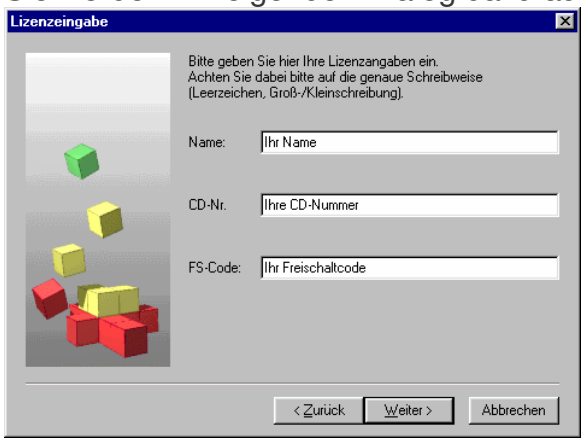
Des Weiteren sollten Sie Grundkenntnisse in einer beliebigen Programmiersprache besitzen. Die Installation und der erste Start müssen mit Administratorrechten durchgeführt werden.

2.1.2 Setup von der SiSy-CD

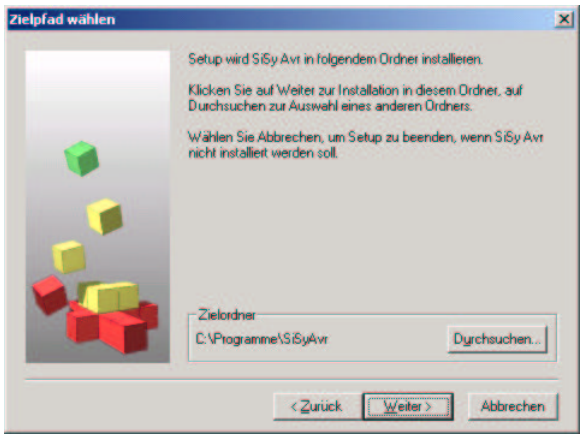
Legen Sie die CD „SiSy“ in Ihr CD-ROM-Laufwerk ein. Falls die CD nicht automatisch startet, wählen Sie bitte im Explorer das CD-ROM-Laufwerk und starten die exe-Datei aus der Wurzel des Laufwerks.
Auf dem Startbildschirm stehen Schaltflächen zur Verfügung zum Installieren der Software und zum Öffnen von Begleitdokumenten.
Für die Installation der Software betätigen Sie die entsprechende Schaltfläche. In Abhängigkeit Ihrer Rechnerkonfiguration kann der Start des Setup-Programms einige Sekunden dauern. Das gestartete Setup-Programm wird Sie durch die weitere Installation führen.

Beginn der Installation

Betätigen Sie im Setup-Programm die Schaltfläche „Weiter“. Sie erhalten die Lizenzbestimmungen. Bitte lesen Sie diese sorgfältig durch. Wenn Sie sich mit diesen Bestimmungen einverstanden erklären, bestätigen Sie die Lizenzbestimmungen.
Sie werden im folgenden Dialog dazu aufgefordert, Ihre Lizenzangaben einzugeben.



Danach erscheint folgende Dialogbox. Im unteren Teil wird der Pfad verwaltet, in dem SiSy zu installieren ist. Wenn ein anderer Pfad (bzw. ein anderes Laufwerk) gewünscht wird, ist die Schaltfläche „Durchsuchen“ zu betätigen. Eine Dialogbox erscheint; in den Feldern „Laufwerke“ und „Verzeichnisse“ können Laufwerk und Verzeichnis festgelegt werden.


Zielpfad wählen
Setup wird SiSy Avr in folgendem Ordner installieren.
Klicken Sie auf Weiter zur Installation in diesem Ordner, auf Durchsuchen zur Auswahl eines anderen Ordners.
Wählen Sie Abbrechen, um Setup zu beenden, wenn SiSy Avr nicht installiert werden soll.
Zielordner
C:\Programme\SiSyAvr
Durchsuchen...
< Zurück
Weiter >
Abbrechen

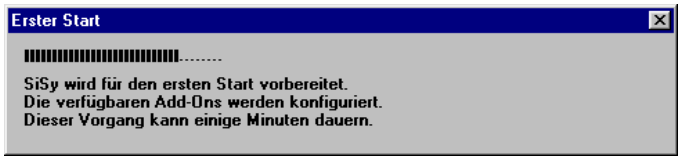
Wählen Sie danach den Programmordner, in dem die Verknüpfungen von SiSy eingefügt werden. Sie können den Zielordner ändern.

Setzen Sie die Installation mit „Weiter“ fort. Der für die Installation erforderliche Speicherplatz wird angezeigt und mit dem auf dem Laufwerk freien Speicherplatz verglichen. Wenn der freie Speicherplatz nicht ausreichend ist, erfolgt eine Warnung und Sie sollten den erforderlichen Speicherplatz zur Verfügung stellen.

Hinweis:

In SiSy sind 2 Dateien enthalten, die Makros beinhalten („handbuch.doc“, „multi.doc"). Von einigen Virenscannern werden diese Makros als „Virus“ erkannt und entsprechend behandelt. In den Heuristik-Einstellungen des Virenscanners kann diese Behandlung unterdrückt werden.

Abschluss der Installation
Die Installation ist erst nach dem ersten Start von SiSy abgeschlossen. Dieser sollte im Anschluss an das Setup ausgeführt werden. Der Vorgang kann einige Minuten dauern.


Erster Start
SiSy wird für den ersten Start vorbereitet.
Die verfügbaren Add-Ons werden konfiguriert.
Dieser Vorgang kann einige Minuten dauern.

Mit Beendigung dieses Vorgangs erscheint auf Ihren Bildschirm der Dialog „Willkommen in SiSy“. Folgen Sie dann den Hinweisen des Assistenten.

2.2 Beschaffen bzw. Herstellen der Hardware

Alle Ausführungen, Übungen und Aufgabenstellungen beziehen sich auf das myAVR Board als Referenzhardware. Wenn Sie Spaß an Elektronik haben, können Sie die Hardware auch selbst fertigen. Die Komponenten erhalten Sie als Bausatz oder fertig bestückt inklusive Schaltplan etc. unter www.myAVR.de.
Ausführliche Beschreibungen zur Programmierung mit Assembler, C/C++, BASCOM sind nicht Inhalt dieses Handbuches. Weiterführende Erklärungen dazu sind im „myAVR Lehrbuch Mikrocontrollerprogrammierung“ enthalten.

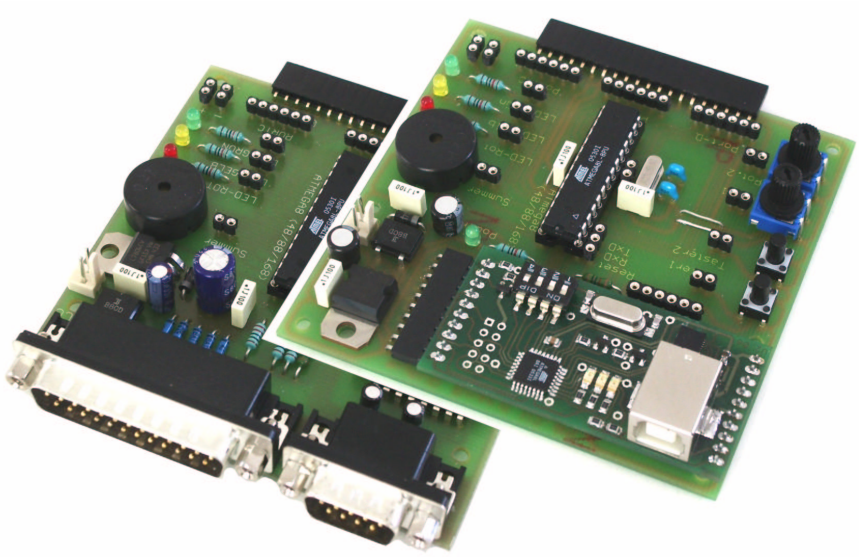


Abbildung 1: myAVR Board mit USB- bzw. LPT-Port

3 Arbeiten mit SiSy AVR, allgemein

Um dem Nutzer zu erläutern, wie er in SiSy modellieren kann, werden zweckentsprechende Definitionen der Begriffe gegeben.

3.1 Projektarbeit

3.1.1 Was ist ein SiSy-Projekt?

Ein SiSy-Projekt ist eine abgegrenzte Menge von verknüpften Elementen für ein zu bearbeitendes Problem. Alle aktuellen Daten sind in einer Projektdatenbank gespeichert. Die Projektdatenbank besteht aus einer Anzahl von Dateien im Projektverzeichnis, wobei jedes Projekt sein eigenes Verzeichnis hat. Durch das Anlegen eines Projektarchivs können diese in einer Datei komprimiert werden.

3.1.2 Neues Projekt erstellen

Ein neues Projekt wird eingerichtet und für die Bearbeitung bereitgestellt. Die Definition (Erstellung) eines neuen Projektes erfolgt durch Vergabe eines Projektdateinamens und/oder durch Überschreiben eines alten Projektes gleichen Namens. Über die Schaltfläche „Verzeichnis“ kann die vorgeschlagene Pfadangabe geändert werden.

Hinweis:
Falls ein bereits vorhandener Projekt- oder Verzeichnisname gewählt wird, erfolgt eine Überschreibwarnung.

Nach der Vergabe eines Projektnamens kann im nachfolgenden Fenster ein Vorgehensmodell ausgewählt werden. Je nach Vorgehensmodell können Vorlagen oder Assistenten das Erstellen eines neuen Projektes unterstützen.

3.1.3 Vorhandenes Projekt öffnen

Ein vorhandenes Projekt wird geöffnet und die abgespeicherten Diagramme und Objekte sowie alle Möglichkeiten für die weitere Bearbeitung werden verfügbar gemacht. Die Wahl des Projektes erfolgt durch Klicken auf den entsprechenden Namen oder über die Schaltfläche „Suchen“.

Beim Aktivieren der Checkbox „Beschreibung anzeigen“ werden zum markierten Projekt Informationen angezeigt. Dazu gehören außer der Beschreibung auch Informationen zum Bearbeitungsstand.

3.1.4 Projekt archivieren

Menü *Projekt/Archiv/Anlegen.*
Es kann ein komprimiertes Archiv des Projektes erzeugt werden. Dies ist besonders aus Gründen der Datensicherheit sinnvoll. Zielverzeichnis und Dateiname für die Archivdatei werden vorgeschlagen und können korrigiert werden. Der Umfang der Archivierung ist festzulegen und die Entscheidung für eine Komprimierung zu treffen, wobei diese empfohlen wird. Wenn ein Projekt unter einem bereits vorhandenen Archivnamen angelegt werden soll, wird eine Überschreibwarnung angezeigt. Bei Auswahl von „Nein“ wird die Erstellung des Archivs abgebrochen, bei „Ja“ wird das Projekt archiviert.

Hinweis: SiSy bietet die Möglichkeit des regelmäßigen Abspeicherns verschiedener Arbeitsstände, d.h. ein archiviertes Projekt wird nicht überschrieben. Ein Projektstand kann in einer neuen Archivdatei abgelegt werden

3.1.5 Projektarchiv einlesen

Menü Projekt/Archiv/Einlesen.
Hierunter versteht man das Einlesen eines Archivs zum Zweck der Rekonstruktion des Projektes.
Einlesen bedeutet Entpacken eines archivierten Projektes. Dazu sind der Archivpfad und der Dateiname des Archivs sowie das Zielverzeichnis anzugeben.
Hinweis:
Wenn im Zielpfad des Entpackens bereits ein Projekt existiert, erscheint eine Überschreibwarnung.

3.2 Die Modellierungselemente von SiSy

Werkzeug
SiSy stellt für die Bearbeitung der Modelle und Teilmodelle Werkzeuge der entsprechenden Methodik bereit. Werkzeuge sind Editoren, mit denen in einem Fenster die grafische Darstellung (Diagramme) der Modelle bearbeitet werden kann.

Diagramme
Diagramme sind grafische Darstellungen von Modellen oder Teilmodellen, die mit einem bestimmten Werkzeug erstellt werden. Die Modellierungselemente werden als Objekte in den Diagrammen unter Einhaltung von Regeln zusammengestellt.

Objekte
Objekte sind mögliche Modellelemente in Diagrammen, z.B. „kleines Programm“ in der „Programmierung“. Objekttypen sind konkrete Ausprägungen von Objekten, die in einem Diagramm angelegt wurden, z.B. Objekttyp „Lauflicht“ vom Objekt „kleines Programm“.

Referenzen
Die Objekte eines Diagramms können in anderen Diagrammen wiederverwendet werden. Durch das Hineinziehen aus dem Navigator oder aus einem offenen Diagramm wird eine Referenz vom Originalobjekt erzeugt. Die Referenz ist nur ein Verweis auf das Original, alle angezeigten Informationen wie Kurzname, Langname und Beschreibung werden vom Original bezogen. Somit sind die Informationen in allen Referenzen eines Objektes identisch mit dem Original. Dadurch werden Änderungen dieser Informationen automatisch auf alle Referenzen übertragen. Weiterhin ist es möglich, diese Referenzierung über einen sogenannten Report auszuwerten.

Kanten
Kanten sind Verbindungselemente zwischen Objekten. Eine Verbindung wird durch Ziehen mit der Maus (linke Maustaste) vom Verteiler des selektierten Objektes auf das gewünschte Objekt erreicht. Nach Loslassen der Maustaste und Prüfung der Verbindungszulässigkeit durch SiSy erscheint ein Kanten-Dialog, in dem das Element definiert und individuelle Einstellungen getroffen werden können.
Hinweis:
Bei Verbindung mit gehaltener STRG-Taste wird die Prüfung vernachlässigt und eine Zwangsverbindung erreicht.

Rahmen
Ein Rahmen fasst ausgewählte Objekte des Diagramms optisch zusammen. Er besitzt einen Kurz- sowie Langnamen und eine Objektbeschreibung, kann also als Objekt aufgefasst werden.
Hinweis:
Inhalte von Rahmen sind in Reports oder einer Dokumentengenerierung auswertbar.

3.3 Die Fenster für die Modellierung

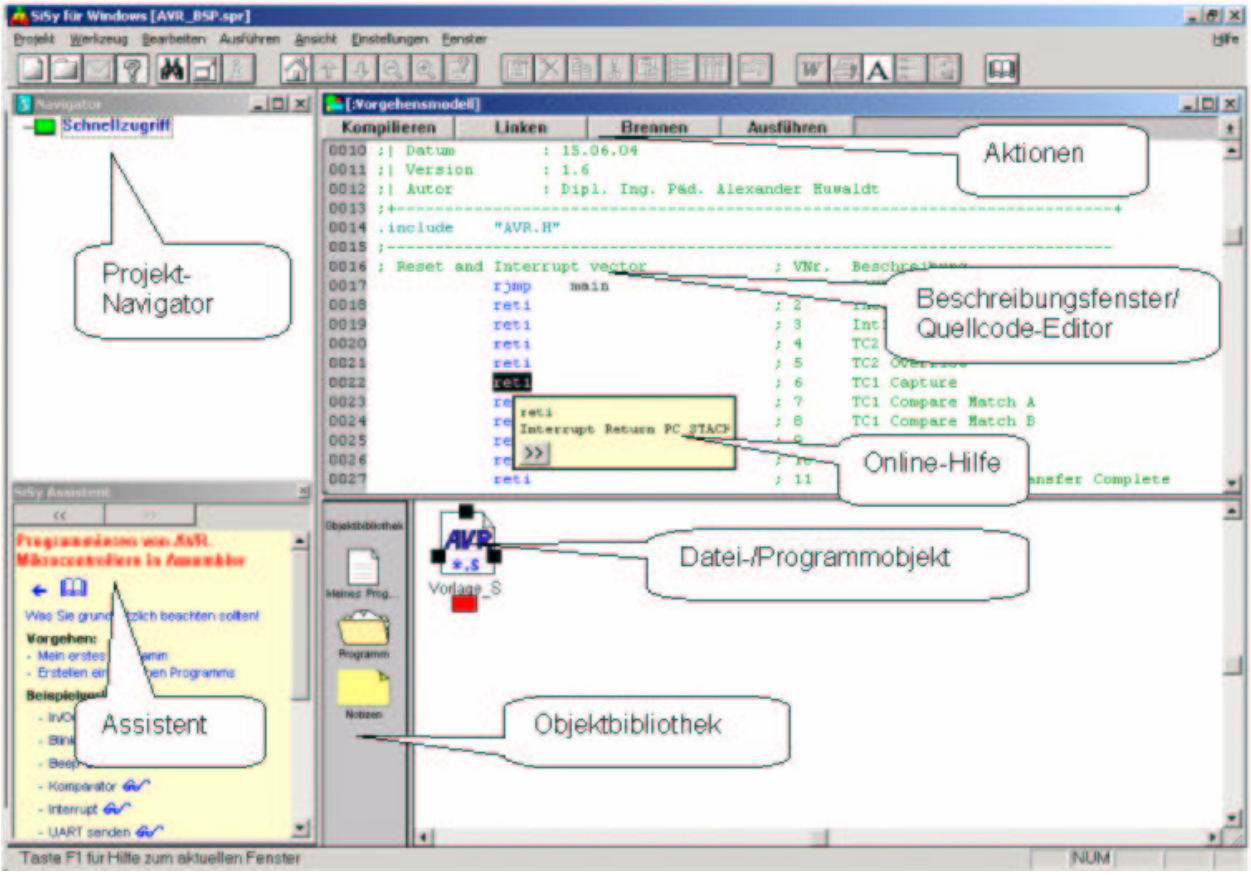


Abbildung 2

Navigator
Dieser befindet sich im linken, oberen Teil des Fensters. Er ermöglicht dem Anwender die Orientierung über die in der Projektdatenbank gespeicherten Objekte sowie deren Bereitstellung für die weitere Verwendung. Nach dem Start von SiSy werden neben dem Vorgehensmodell eine Reihe leicht zu handhabender Schaltflächen, Menüs und weitere Hilfsmittel angezeigt.

SiSy-Assistent
Am linken unteren Bildschirmrand befindet sich diese Nutzerunterstützung.

- Er begleitet den Anwender durch das gesamte Projekt und hält immer passende Informationen zur aktuellen Sicht parat
- Er ist auf die jeweilige Ausgabe von SiSy bezogen.
- Oft können Beispiele als Vorlagen vom Assistenten geladen werden.

Diagrammfenster
Am oberen Bildrand befinden sich das Hauptmenü, das hilfreiche Funktionen zu SiSy bereithält, und eine Werkzeugleiste, mit deren Hilfe schnell auf nützliche Anwendungen zugegriffen werden kann. Das Diagrammfenster nimmt die rechte Bildschirmseite ein und ist der Raum, in dem der Nutzer modelliert. Es enthält:

- das ausgewählte Vorgehensmodell,
- die Objektbibliothek mit den möglichen Objekten des aktuellen Diagramms sowie
- ein Fenster zur Beschreibung des markierten Objekts, in diesem Fall zum Editieren des Quelltextes.

Die Bedienelemente/Objektbibliothek
SiSy bietet, wie bei Windows-Anwendungen üblich, die Steuerung von Befehlen über das Hauptmenü, über die Werkzeugleisten, die Tastatur oder die Objektbibliothek an. Darüber hinaus enthalten das Kontextmenü und der Navigator Steuerfunktionen.

Die Anzahl der möglichen Befehle in der Menüleiste ist abhängig davon, ob ein Projekt geöffnet ist. Ist das nicht der Fall, erscheint ein Menü mit wenigen Befehlen. Bei einem geöffneten Projekt hält SiSy umfangreichere Menüs bereit. Die wichtigsten Menübefehle befinden sich auch als grafische Schaltfläche in der Werkzeugleiste, die eine schnelle und effiziente Bedienung des Programms ermöglicht. Die Toolbox-Darstellung bietet dem Anwender wichtige Programmfunktionen als direkten Link an.
Ein weiteres Bedienelement ist die Objektbibliothek. Sie unterstützt das Anlegen neuer Objekte.

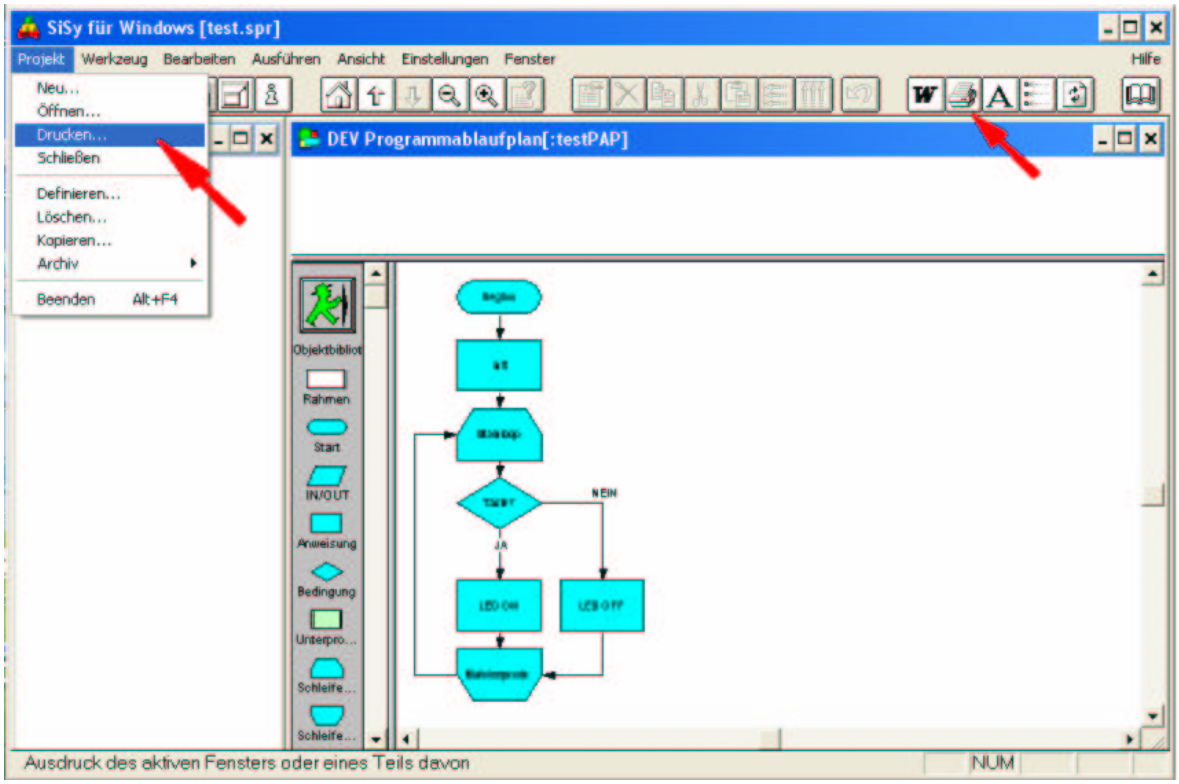
3.4 Druckfunktionen in SiSy

Sie haben in SiSy verschieden Möglichkeiten Projektinhalte wie Grafiken, Übersichten, Quellcodes oder ganze Projektdokumentationen zu drucken. Dabei ist zu berücksichtigen, dass in SiSy bestimmte Informationen wie zum Beispiel die Darstellung eines Programmablaufplanes sichtbare Elemente eines Diagramms sind und andere Teile wie zum Beispiel der Quellcode eines Elementes nur über Dialoge oder bei Selektierung des Elementes sichtbar sind. Je nachdem, welchen Inhalt Sie dokumentieren wollen, richtet sich die Auswahl der betreffenden Druckfunktion.

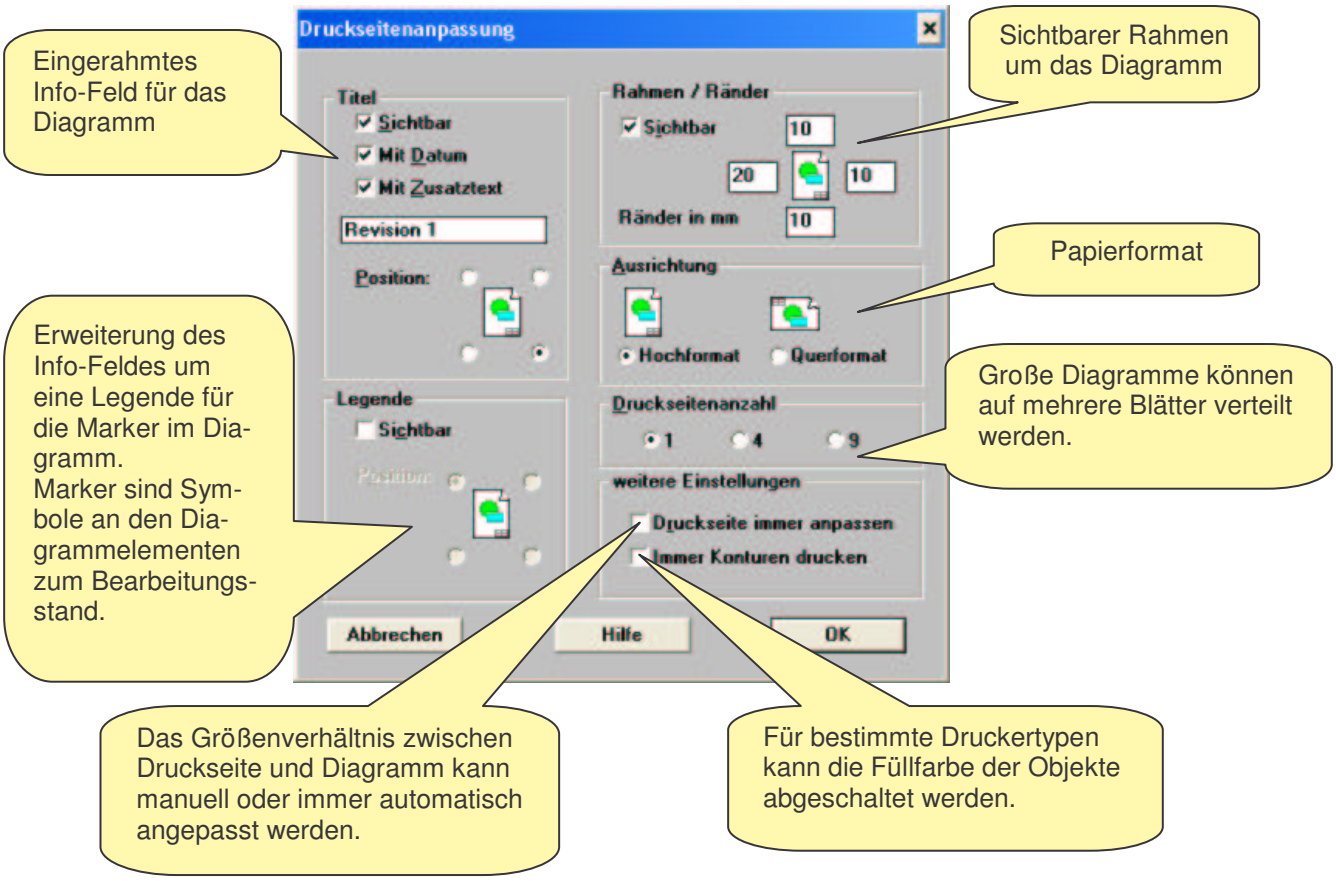
3.4.1 Diagramme drucken

Wenn Sie ein einzelnes Diagramm, also den sichtbaren Inhalt eines Diagrammfensters drucken wollen, gehen Sie wie folgt vor:

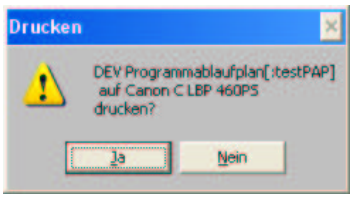
- ggf. Projekt öffnen
- das gewünschte Diagramm öffnen
- die Menüfolge *Projekt/Drucken* wählen oder das Druckersymbol in der Werkzeugleiste aktivieren



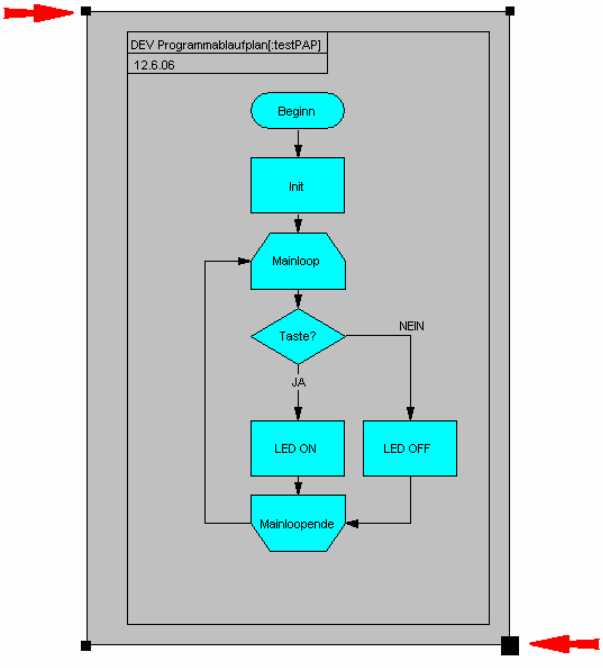
- Sie erhalten den Dialog zum Einrichten der Druckseite, wählen Sie die gewünschten Optionen aus!



- Zum Drucken wählen Sie die Schaltfläche OK. Der Ausdruck erfolgt auf dem Standarddrucker des Systems



- Im Hintergrund ist die Druckvorschau zu sehen. Der Druckvorgang kann hier abgebrochen werden, um die Einstellungen zu überarbeiten. Dabei kann die relative Position und das Größenverhältnis der Druckseite zum Diagramm verändert werden. Der Dialog zum Verändern der Einstellungen lässt sich per Doppelklick auf den Selektierungsmarken der Druckseite öffnen.



- Die Druckseitenansicht lässt sich über die Menüfolge *Ansicht/Druckseite* ein- und ausblenden

- Der Druckvorgang kann über das Druckersymbol in der Werkzeugleiste jederzeit gestartet werden.

3.4.2 Grafiken und Inhalte drucken (QuickDok)

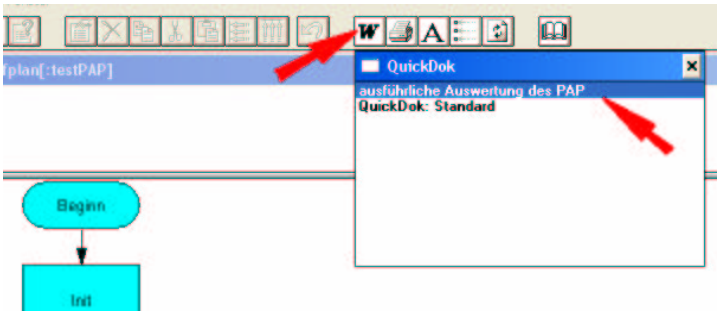
Viele Projektinformationen sind kein sichtbarer Teil von erstellten Diagrammen. Diese wurden über Dialoge und Masken eingegeben und stehen als Attribute in der Projektdatenbank zur Verfügung. Um diese Informationen auszudrucken, stellt SiSy für jeden Diagrammtyp eine Reportfunktion zur Verfügung, mit der die wesentlichen Informationen, Inhalte und Attribute des Diagramms und der Objekte in einem Diagramm, als Word-Dokument generiert werden.

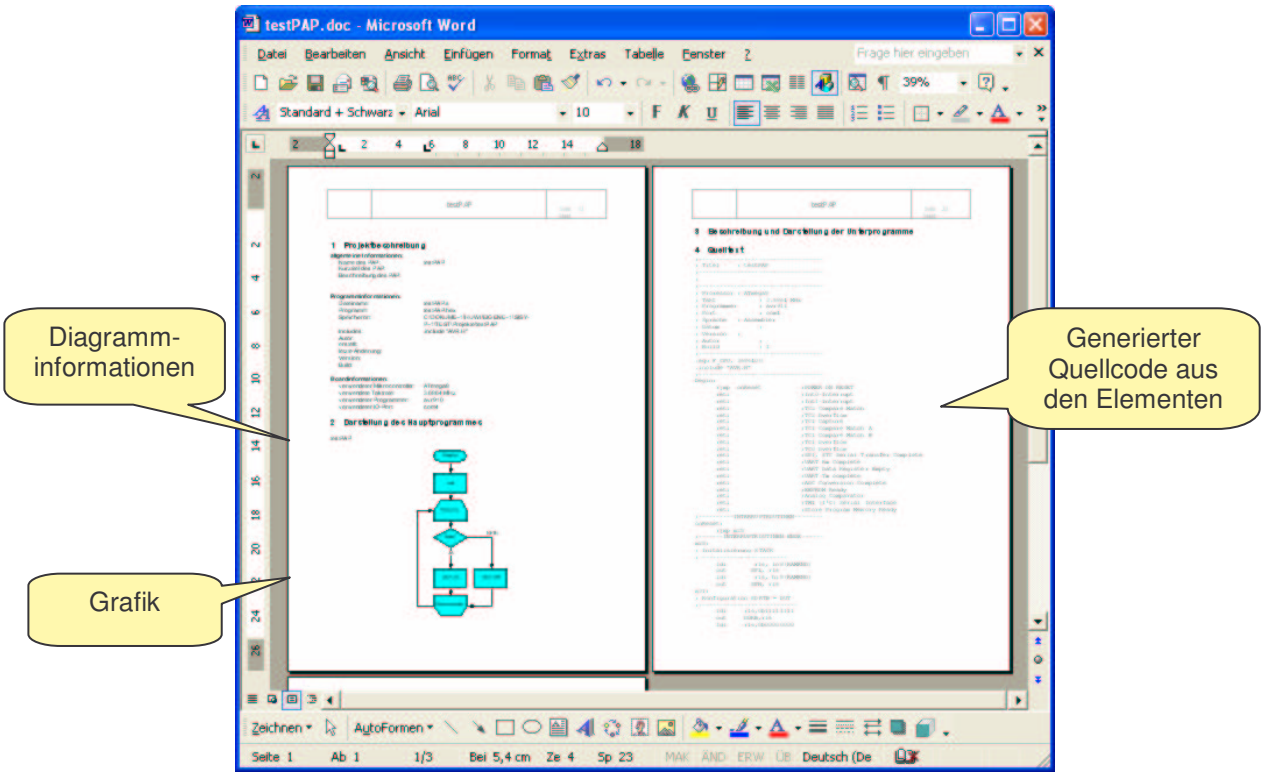
Für das Generieren des Word-Dokumentes muß auf dem PC MS Word installiert sein. Unterstützt werden die Versionen Word 2000/XP/2003.
Die Dokumentengenerierung basiert auf Makros; deshalb ist in Word der Makrovirenschutz auf *mittel* zu stellen.
Vorgehensweise in Word:

- Wählen Sie die Menüfolge *Extras/Makro/Sicherheit*
- Aktivieren Sie auf der Registerkarte „Sicherheitsstufe“ *mittel*
- Auf der Registerkarte „Vertrauenswürdige Quellen“ setzen Sie die Häkchen bei „Allen Vorlagen und Add-Ins vertrauen“ und wenn vorhanden bei „Zugriff auf Visual Basic Projekte vertrauen“

Die Reportfunktion zum Generieren des Word-Dokumentes aktivieren Sie über das Symbol „QuickDok“ in der Werkzeugleiste.

- Symbol (QuickDok) in der Werkzeugleiste wählen
- Aus der Liste der angebotenen Reportfunktionen auswählen
- Dateinamen für den Report vergeben
- Makros aktivieren

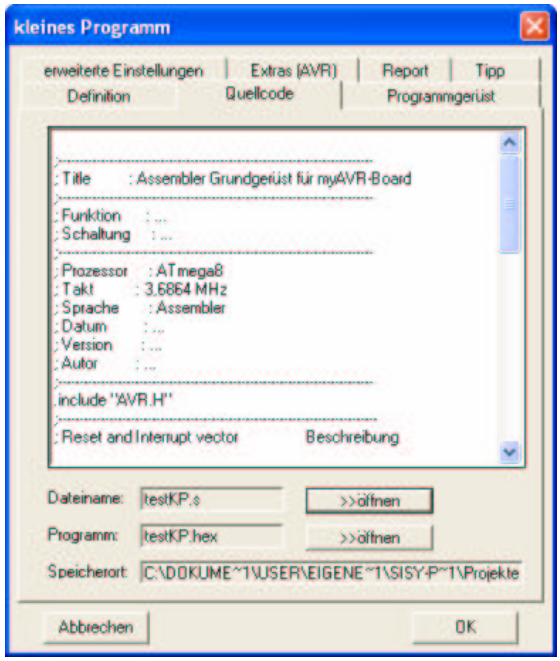




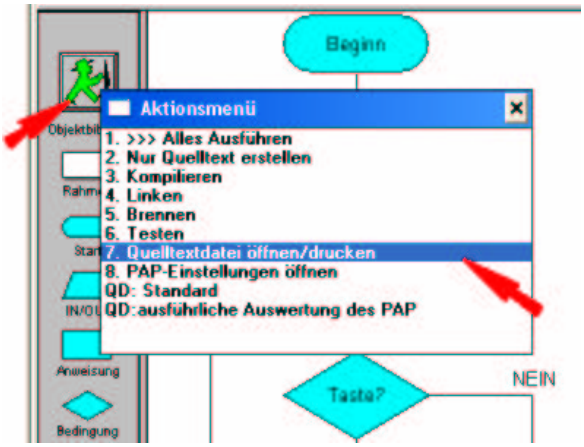
3.4.3 Nur Quellcodes drucken

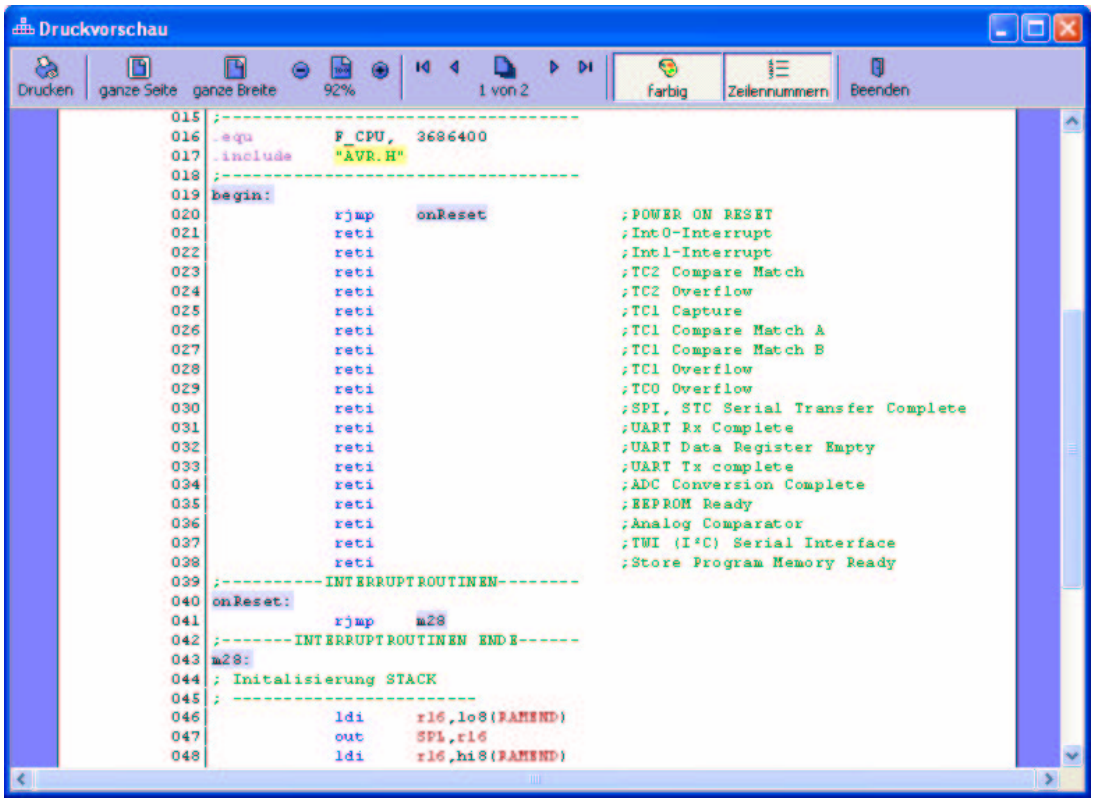
Für das Ausdrucken von Quellcodes bietet SiSy einen speziellen Quellcode-Druckassistenten. Damit ist es möglich Quellcodes formatiert, in Syntaxfarben und mit Zeilennummern auszudrucken.

- kleines Programm, Quellcode drucken
  - kleines Programm selektieren
  - rechte Maustaste „Definieren…“
  - Dialogseite „Quellcode“
  - Schaltfläche „>>öffnen“



- Programmablaufplan, Quellcode drucken
  - PAP öffnen
  - Ampelmännchen aktivieren
  - „Quellcodedatei öffnen/drucken“

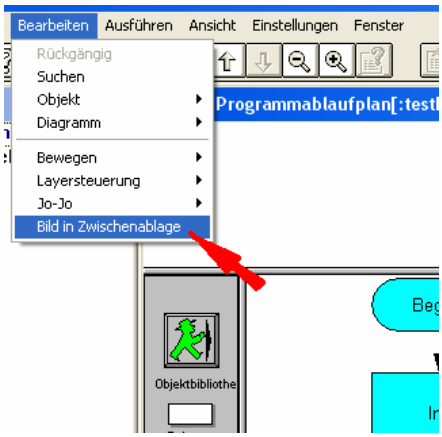


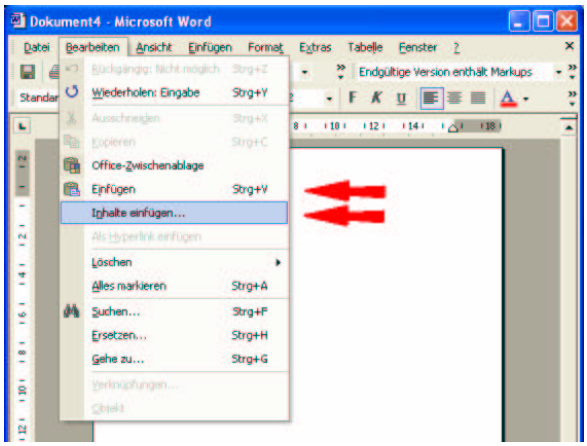


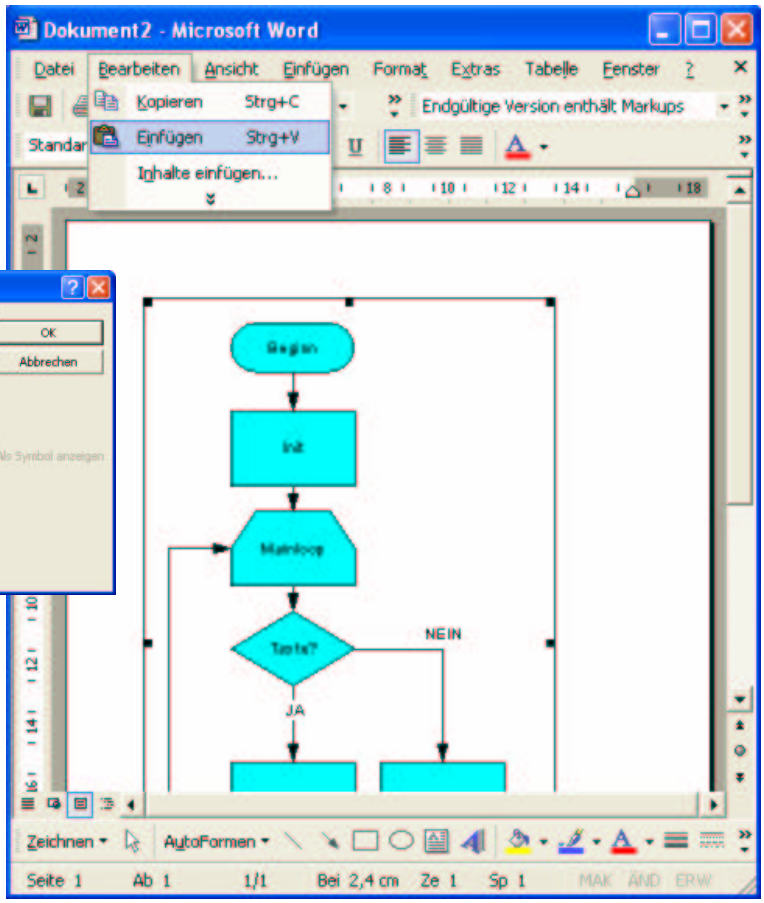
3.4.4 Nutzen der Zwischenablage

Oft ist es erforderlich in Projektdokumentationen die Diagramme als Bilder einzufügen. In SiSy werden die Diagramme nicht als Bilder gespeichert sondern zur Laufzeit aus den Modellinformationen generiert. Um die Bilder der Diagramme weiter zu verwenden, steht dem Anwender die Funktion „Bild in Zwischenablage“ zur Verfügung. Dabei erstellt SiSy eine skalierbare Vektorgrafik (WMF) und legt diese in die Zwischenablage (Copy). Die Grafik kann nun von anderen Anwendungen über den Befehl „Einfügen“ (Past) beliebig weiter verwendet werden.

- gewünschtes Diagramm öffnen
- Menüfolge *Bearbeiten/Bild in Zwischenablage* wählen
- Zielanwendung, zum Beispiel Word öffnen
- Menüfolge *Bearbeiten/Einfügen* oder *Bearbeiten/Inhalte einfügen* wählen
- Gegebenfalls einzufügendes Grafikformat wählen







4 Die Hilfefunktionen in SiSy

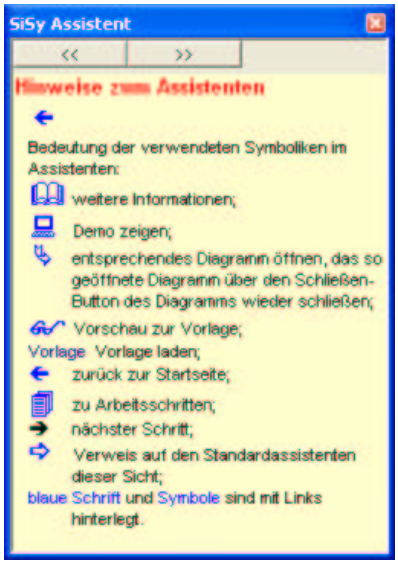
Nutzen Sie die zahlreichen Hilfen und Vorlagen, die SiSy dem Entwickler bietet!

4.1 Der Assistent

Der Assistent ist hilfreich bei der Unterstützung und Führung des Nutzers im Umgang mit SiSy. Er befindet sich standardmäßig im linken, unteren Bildschirmbereich. Der Assistent kann entweder über das Menü *Ansicht/Assistent* oder über die Werkzeugleiste geöffnet werden, falls dies nicht beim Programmstart erfolgte. Der Assistenten begleitet Sie im gesamten Projekt. Sie erhalten immer passende Informationen zum aktuellen Diagrammtyp und haben die Möglichkeit, durch Links weitere Hilfethemen aufzurufen oder Vorlagen in Ihr Projekt zu laden.

Beachte: Der Assistent ist auf die jeweilige Ausgabe von SiSy, die verfügbaren Add-Ons und das gewählte Modell bezogen.

Bedeutung der verwendeten Symboliken im Assistenten:


SiSy Assistent
Hinweise zum Assistenten
Bedeutung der verwendeten Symboliken im Assistenten:
weitere Informationen;
Demo zeigen;
entsprechendes Diagramm öffnen, das so geöffnete Diagramm über den Schließen-Button des Diagramms wieder schließen;
Vorschau zur Vorlage;
Vorlage laden;
zurück zur Startseite;
zu Arbeitsschritten;
nächster Schritt;
Verweis auf den Standardassistenten dieser Sicht;
blaue Schrift und Symbole sind mit Links hinterlegt.

&	weitere Informationen anzeigen; öffnet eine Hilfedatei (*.chm, *.hlp, *htm)
:	Demovideo zur Handhabung zeigen; öffnet eine Animation oder Videomitschnitt der Bildschirmarbeit (AVI, ScreenCam- oder FLASH-Film)
~	entsprechendes Diagramm öffnen, das so geöffnete Diagramm kann über die Schließen-Schaltfläche des Diagramms wieder geschlossen werden;
Beispiel 1	eine Diagrammvorlage laden;
$	Vorschau zur Diagrammvorlage;
ç	zurück zur Startseite des Assistenten;
4	ein kleines Skript zu Arbeitsschritten anzeigen;
Ł	nächster Schritt (Arbeitsschritt);
ð	Verweis auf den Standardassistenten dieser Sicht;
blaue Schrift	und
blaue Symbole	sind mit Links hinterlegt und können per Mausklick aktiviert werden.

Beispiele für Assistenten:

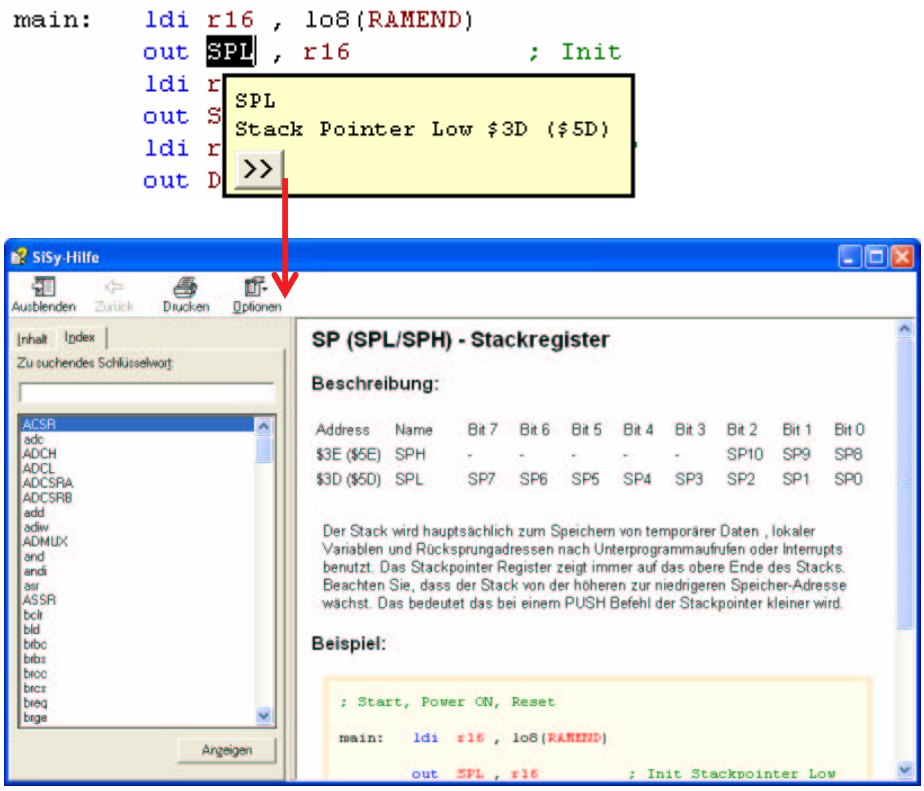
SiSy Assistent
Vorgehensmodell AVR
Das Vorgehensmodell AVR bietet verschiedene Möglichkeiten zur Entwicklung von AVR-Programmen:
- kleines Programm
- Programmablaufplan (PAP)
- UML-Klassendiagramm
Jedes dieser Diagramme bietet Ihnen nach der Modellierung o.Ä. Quelltexteingabe die Möglichkeit
- zur Generierung
- zum Debuggen sowie
- zum Brennen
des Programmes.
- Wissenswertes über SiSy
- zur SiSy Homepage
- Add-Ons für SiSy
SiSy Assistent
Programmieren mit SiSy
In diesem Diagramm können Sie Programme in verschiedenen Programmiersprachen erstellen.
- AVR-Assembler
- AVR-C
Hinweise zum Debugger
Hinweise zum allgemeinen Vorgehen beim Erstellen eines ...
... kleinen Programms
... Programms
Weitere Hinweise zu SiSy finden Sie hier:
- Wissenswertes über SiSy
SiSy Assistent
Programmieren von AVR-Mikrocontrollern in Assembler
Vorgehen:
- Was Sie grundsätzlich beachten sollten!
- Mein erstes Programm
Beispielvorlagen:
1 Grundgerüste
2 Beispielprogramme
Hinweis:
Mit
- der Taste "F4" oder
- mit dem Symbol ± in der rechten oberen Ecke des Quelltextfensters
kann das Quelltextfenster vergrößert bzw.

4.2 Die Online-Hilfe

Bei der Eingabe von Quellcodes im dafür vorgesehenen Editorfenster werden reservierte Worte (Bezeichner, Schlüsselworte) der gewählten Programmiersprache durch verschiedenfarbiges Einfärben (Syntaxfarben) hervorgehoben. Zu den hervorgehobenen Bezeichnern existiert in der Regel auch eine kurze Online-Hilfe und eine ausführlichere Hilfe. Die Online-Hilfe ist ein Pop-Up-Fenster, welches bei der Eingabe des Bezeichners automatisch eingeblendet wird. In dem Pop-Up ist eine kurze Hilfestellung zu dem Bezeichner eingeblendet. Die Online-Hilfe ist auch durch Doppelklick auf dem Bezeichner erreichbar. Steht eine Hilfe mit ausführlicheren Informationen zur Verfügung, wird in dem Pop-Up eine Schaltfläche [>>] zum Öffnen des Hilfethemas angeboten.

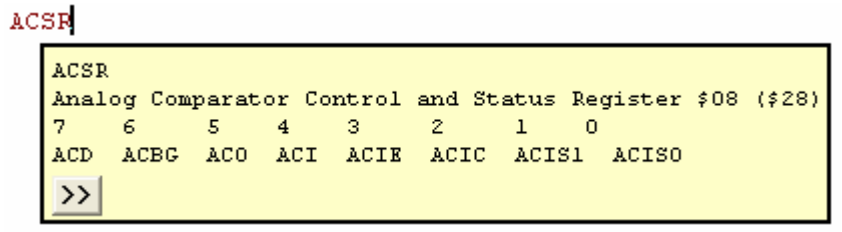
Register-Hilfe

Bei der Eingabe von bekannten Registernamen wird die Bezeichnung des Registers und dessen Adresse eingeblendet. Die Schaltfläche [>>] öffnet eine Hilfedatei mit dem entsprechenden Hilfethema zu dem eingegeben Register.


main: ldi r16 , lo8(RAMEND)
out SPL , r16 ; Init
SPL
Stack Pointer Low $3D ($5D)
>>
SiSy-Hilfe
Ausblenden
Zurück
Drucken
Optionen
Inhalt
Index
Zu suchendes Schlüsselwort
Anzeigen
SP (SPL/SPH) - Stackregister
Beschreibung:
Address Name Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0
$3E ($5E) SPH - - - - - SP10 SP9 SP8
$3D ($5D) SPL SP7 SP6 SP5 SP4 SP3 SP2 SP1 SP0
Der Stack wird hauptsächlich zum Speichern von temporärer Daten , lokaler Variablen und Rücksprungadressen nach Unterprogrammaufrufen oder Interrupts benutzt. Das Stackpointer Register zeigt immer auf das obere Ende des Stacks. Beachten Sie, dass der Stack von der höheren zur niedrigeren Speicher-Adresse wächst. Das bedeutet das bei einem PUSH Befehl der Stackpointer kleiner wird.
Beispiel:
; Start, Power ON, Reset
main: ldi r16 , lo8(RAMEND)
out SPL , r16 ; Init Stackpointer Low

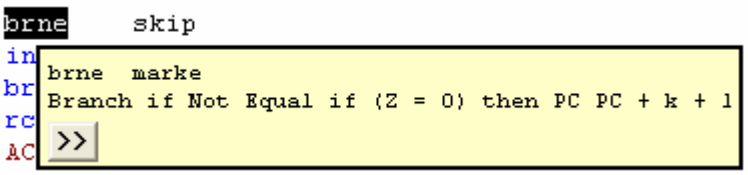
Bit-Hilfe

Wenn für ein Register detaillierte Informationen zu der Bedeutung/Funktion der einzelnen Bits vorliegen, wird zusätzlich im Pop-Up eine Kurzreferenz der Bits angezeigt. Die Schaltfläche **[>>]** öffnet eine Hilfedatei mit dem entsprechenden ausführlichen Hilfethema.



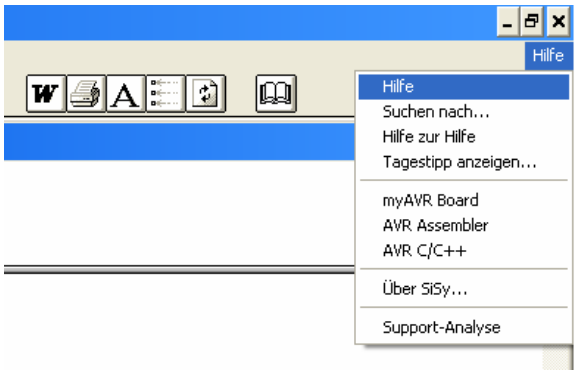
Befehlshilfe

Bei der Eingabe von Befehlen wird in der Regel die Bedeutung bzw. Funktion des Befehls und ein kurzes Syntaxbeispiel eingeblendet. Die Schaltfläche **[>>]** öffnet eine Hilfedatei mit dem entsprechenden ausführlichen Hilfethema.



4.3 Die allgemeine Hilfe

SiSy bietet neben der direkten Hilfe bei der Eingabe von Schlüsselworten auch allgemeine Hilfen an.



In SiSy AVR sind das zum Beispiel die allgemeine Hilfe zum Modellierungstool SiSy und die Hilfen zum myAVR Board, dem AVR Assembler und AVR C. Die Hilfedateien verfügen über einen Index sowie die Möglichkeit der Volltextsuche.

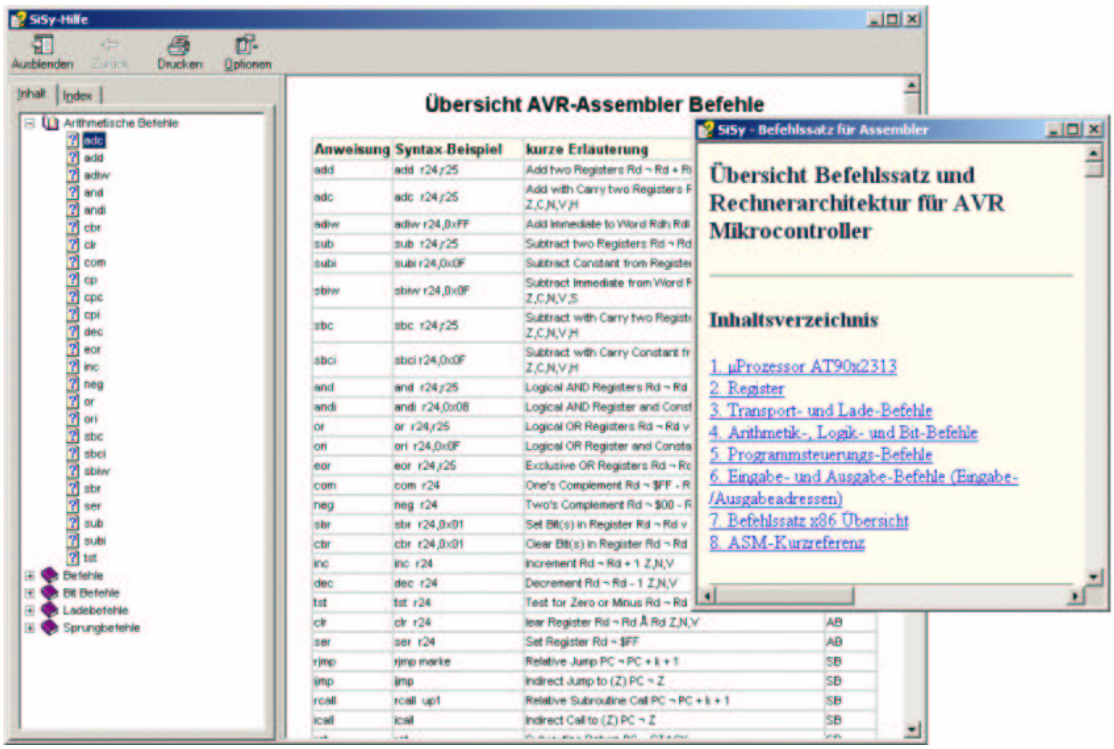


Abbildung 3: SiSy AVR-Assembler Hilfe

5 Entwicklung eines kleinen Programms mit SiSy AVR

Schauen wir uns als nächstes kurz in der Entwicklungsumgebung SiSy AVR um. Für die Entwicklung von Mikrocontrollerlösungen bietet sich für den Einstieg die einfache Programmierung (kleines Programm) an.

5.1 Voraussetzungen

Für die Bearbeitung der folgenden Aufgaben benötigen Sie die aufgeführte Software und Hardware.

Software:

- SiSy AVR ab Version 2.17 oder
- SiSy Ausgabe Developer, Professional oder BS und das installierte Add-On AVR

Hardware:

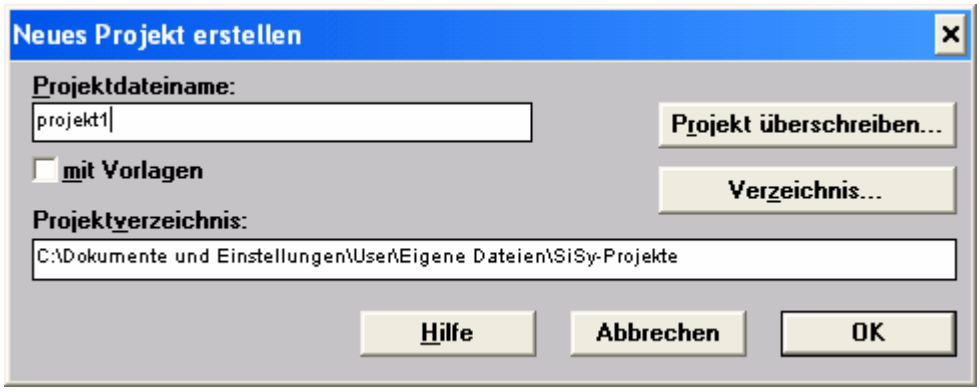
- Ein bestücktes myAVR Board
- Programmierkabel (USB bzw. LPT)
- eventuell Nullmodemkabel für die COM Schnittstelle
- 9 V Batterie bei Bedarf (z.B.: autonomer Einsatz)
- Patchkabel

5.2 Zielstellung

Der Schnelleinstieg zur Mikrocontroller-Programmierung soll Ihnen helfen, SiSy AVR kennen zu lernen und erste Schritte in der hardwarenahen Programmierung mit SiSy zu gehen.
Zielstellung für das erste Beispielprogramm: Die drei LED's auf dem myAVR Board sollen nacheinander aufleuchten und damit ein „Lauflicht" erzeugen.

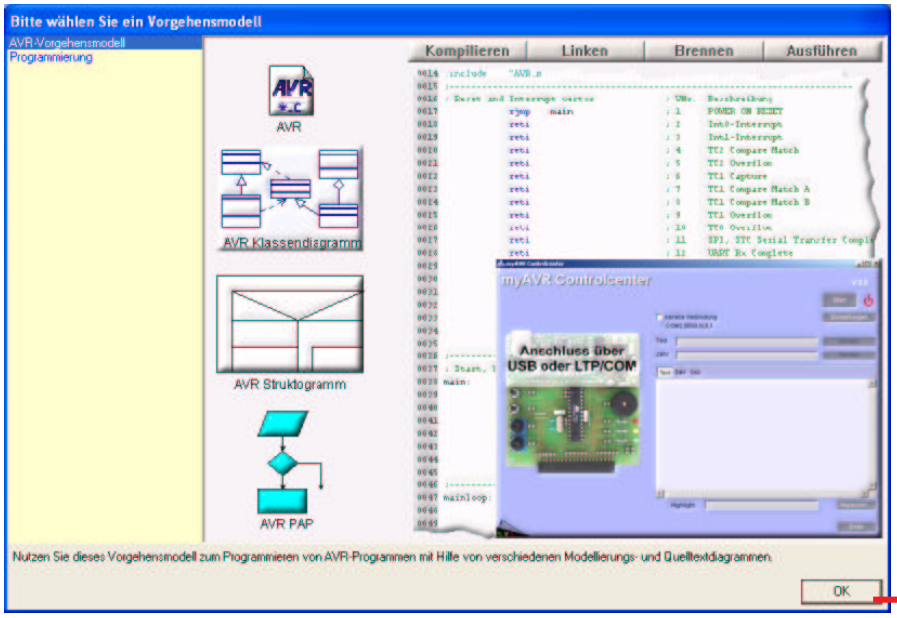
5.3 Vorgehen

Ein neues Projekt anlegen
Starten Sie SiSy, wählen Sie Assistent öffnen und bestätigen Sie mit OK. Wählen Sie den Menüpunkt *Neues Projekt anlegen*, vergeben Sie den Projektnamen „Lauflicht". Wählen Sie das Vorgehensmodell „Programmierung". Wählen Sie aus dem Dialog der Diagrammvorlagen „keine Vorlage verwenden" oder brechen Sie den Dialog ab.

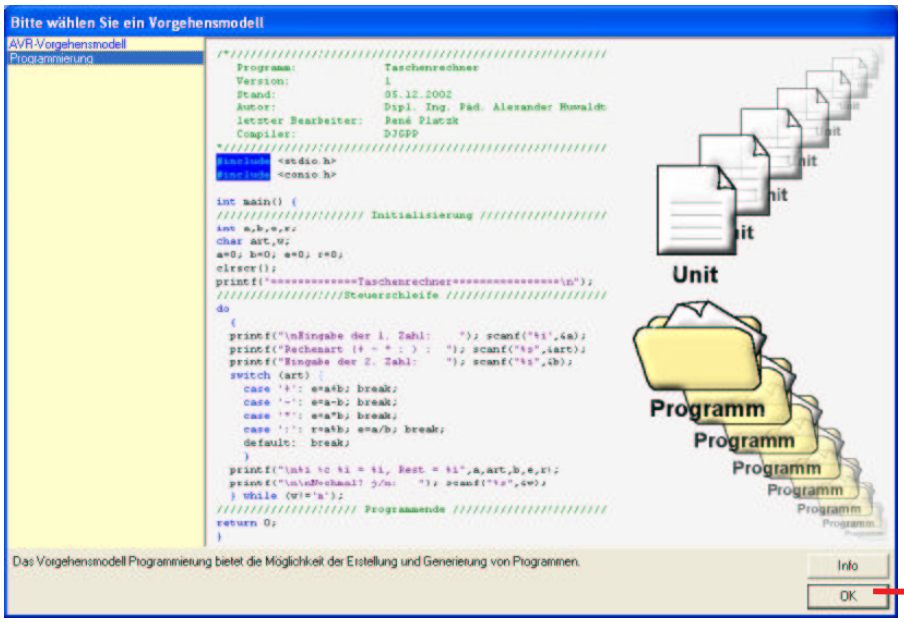


Hinweis:
In SiSy legen Sie stets ein Projekt an. In dieses Projekt integrieren Sie Ihr Programm bzw. mehrere Programme. Unabhängig vom Programmnamen benötigt jedes Projekt einen Namen.

Wenn in ihrer SiSy Ausgabe mehrere Vorgehensmodelle (Hauptebenen) zur Verfügung stehen, wird eine Auswahl der Hauptebene angeboten. Mit der Auswahl des Vorgehensmodells (VGM) entscheiden Sie über das Profil des Projektes. Wenn ein Projekt klein und die Modellierung von Programmablaufplänen nicht erforderlich ist, wählt man das einfachere Vorgehensmodell; in dem gezeigten Beispiel „Programmierung“. Bei komplexen Projekten, für die neben kleinen Programmen auch Programmablaufpläne, Struktogramme oder Klassendiagramme nötig sind, wird das komplexere Vorgehensmodell „AVR-Vorgehensmodell“ ausgewählt.
Die Auswahl des Vorgehensmodells bestimmt im weiteren Projektverlauf die von SiSy zur Verfügung gestellten Werkzeuge. Bei Auswahl des einfachen Vorgehensmodells „Programmierung“ stehen dann zum Beispiel keine grafischen Werkzeuge wie der Programmablaufplan zur Verfügung. Damit sind die Menüs und Objektbibliotheken entsprechend übersichtlicher.



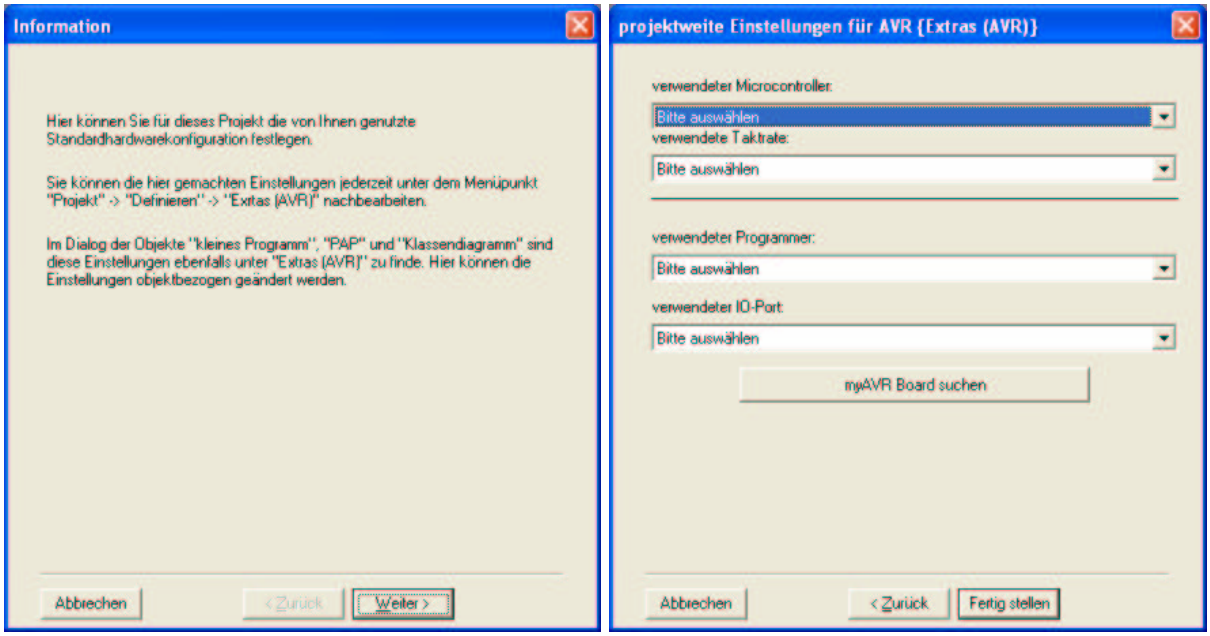
Verfügbare Werkzeuge
- kleines Programm ASM
- kleines Programm C
- großes Programm ASM
- großes Programm C
- Programmablaufplan ASM
- Struktogramm C
- Klassendiagramm C++



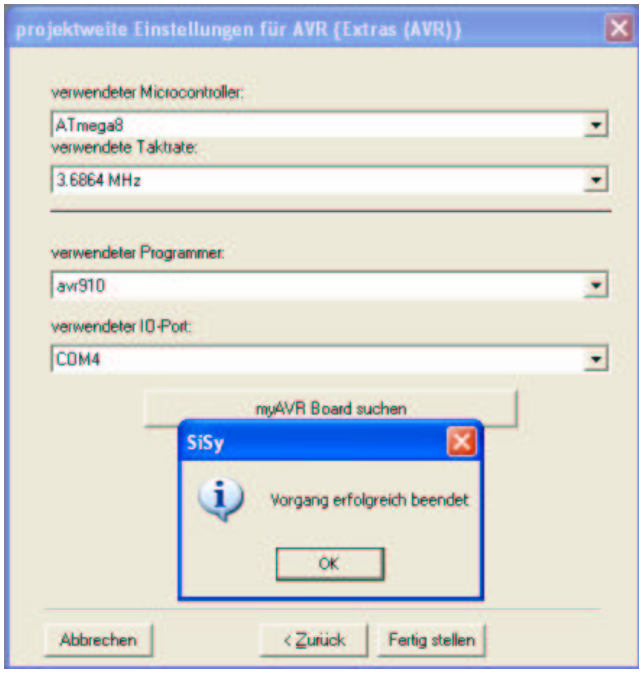
Verfügbare Werkzeuge
- kleines Programm ASM
- kleines Programm C
- großes Programm ASM
- großes Programm C

Im weiteren Verlauf werden entsprechend der Auswahl des Vorgehensmodells unterschiedliche Hilfen zum Erstellen eines Programms angeboten. Für Einsteiger empfiehlt

sich zuerst die Nutzung des Vorgehensmodells „Programmierung“. Bei jedem neuen Projekt müssen die Grundeinstellungen zur verwendeten Zielplattform vorgenommen werden (Mikrocontrollertyp, Taktrate, Programmer und IO-Port).


Information
Hier können Sie für dieses Projekt die von Ihnen genutzte Standardhardwarekonfiguration festlegen.
Sie können die hier gemachten Einstellungen jederzeit unter dem Menüpunkt "Projekt" -> "Definieren" -> "Extras (AVR)" nachbearbeiten.
Im Dialog der Objekte "kleines Programm", "PAP" und "Klassendiagramm" sind diese Einstellungen ebenfalls unter "Extras (AVR)" zu finde. Hier können die Einstellungen objektbezogen geändert werden.
Abbrechen
< Zurück
Weiter >
projektweite Einstellungen für AVR (Extras (AVR))
verwendeter Microcontroller:
Bitte auswählen
verwendete Taktrate:
Bitte auswählen
verwendeter Programmer:
Bitte auswählen
verwendeter IO-Port:
Bitte auswählen
myAVR Board suchen
Abbrechen
< Zurück
Fertig stellen

Wenn ein myAVR Board verwendet wird, kann dieses automatisch gesucht werden. Die Grundeinstellungen für das gefundene Board werden dann automatisch übernommen.



Werden keine Einstellungen vorgenommen, legt SiSy folgende Standardwerte fest:

- Mikrocontroller: ATmega8
- Taktrate: 3,6864 MHz
- Programmer: SP12
- Port: LPT1

Kleines Assembler-Programm anlegen

Erstellen Sie ein Programm für den AVR-Mikrocontroller, indem Sie per Drag & Drop aus der Objektbibliothek ein „kleines Programm“ in das Diagrammfenster ziehen. Markieren Sie das Objekt und wählen Sie aus dem Kontextmenü (rechte Maustaste) *Definieren*.
Auf der Registerkarte „Definition“ tragen Sie den Programmnamen ein (im Beispiel „Lauflicht“) und wählen die Programmiersprache aus, hier „AVR Assembler“; siehe Abbildung 4.

Abbildung 4: Definition kleines Programm

Zielplattform und Programmer einstellen

Kontrollieren Sie auf der Registerkarte „Extras (AVR)“ den ausgewählten Mikrocontroller, siehe Abbildung 5. Die Option „Vorgaben benutzen“ überträgt automatisch die Grundeinstellungen (Mikrocontrollertyp, Taktrate, Programmer und IO-Port) des Projektes in die lokalen Einstellungen. Sollen die lokalen Einstellungen unter „Extras (AVR)“ von den Projekteinstellungen abweichen, muss die Option „Vorgaben benutzen“ abgeschaltet werden.

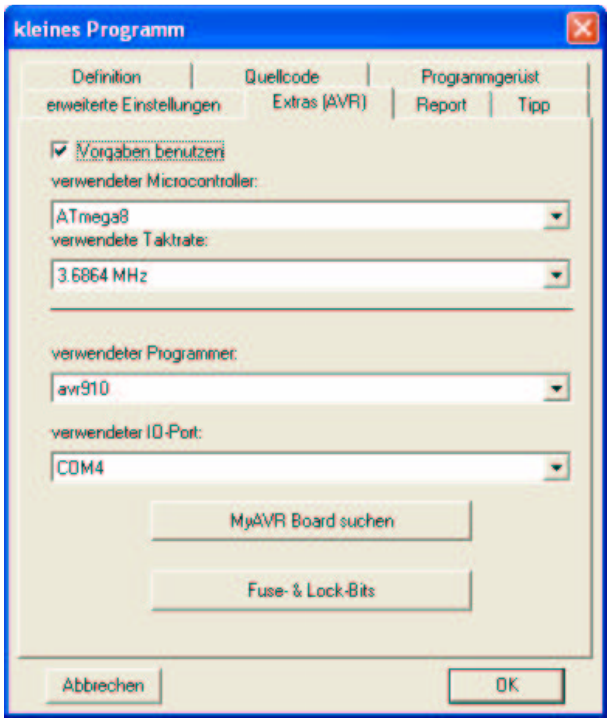


Abbildung 5: Einstellungen Extras (AVR)

Beachte folgende Einstellungen:

- myAVR-Board 1 LPT
  - Programmer: SP12
  - Port: LPTx (z.B. LPT1)
- myAVR-Board 2 USB / mySmartUSB
  - Programmer: AVR910
  - Port: COMx (z.B. COM3)
- mySmartUSB mit Firmware ab 1.5
  - Programmer: AVR911
  - Port: COMx (z.B. COM3)

Programmgerüst laden, Quellcode erstellen

Über die Registerkarte „Programmgerüst“ können Sie das folgende Grundgerüst für ein AVR Assemblerprogramm laden; in der Registerkarte „Quellcode“ können Sie den Quellcode eigenständig eintragen.

Hinweis:

Den Quellcode können Sie auch im Beschreibungsfenster/Editorfenster der SiSy-Benutzeroberfläche eintragen bzw. korrigieren.

Laden Sie die Vorlage für das Programmgerüst oder erstellen Sie den folgenden Quellcode.
Vergleichen Sie dazu auch den Abschnitt zum myAVR Code-Wizard (Abschnitt 7). Das Programmgerüst darf erst geladen werden, wenn der Zielcontroller ausgewählt wurde; die Vorlagen sind controllerspezifisch.

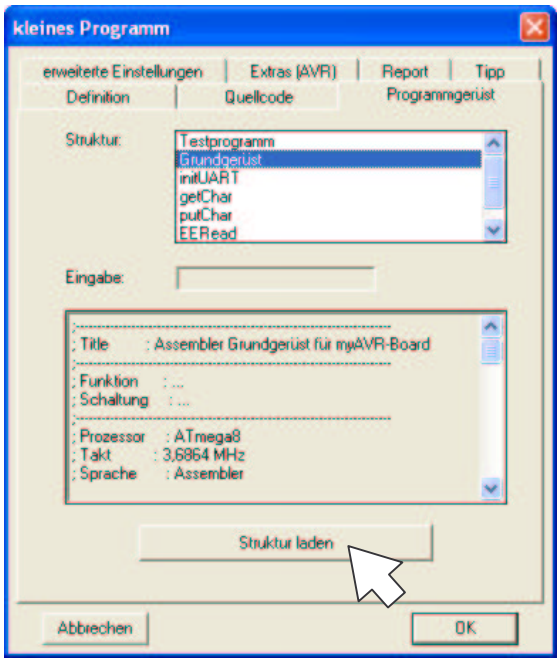


Abbildung 6: Grundgerüst laden

```
;---------------------------------------------------------------
;* Titel            :Programmvorlage für myAVR Board
;* Prozessor        :ATmega8 mit 3,6864 MHz
;* Schaltung        :...
;* Datum            :15.03.2004
;* Autor            :Dipl. Ing. Päd. Alexander Huwaldt
;---------------------------------------------------------------
.include     "avr.h"
;---------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:       rjmp    main       ; 1    POWER ON RESET
             reti               ; 2    Int0-Interrupt
             reti               ; 3    Int1-Interrupt
             reti               ; 4    TC2 Compare Match
             reti               ; 5    TC2 Overflow
             reti               ; 6    TC1 Capture
             reti               ; 7    TC1 Compare Match A
             reti               ; 8    TC1 Compare Match B
             reti               ; 9    TC1 Overflow
             reti               ; 10   TC0 Overflow
             reti               ; 11   SPI, STC Serial Transfer Complete
             reti               ; 12   UART Rx Complete
             reti               ; 13   UART Data Register Empty
             reti               ; 14   UART Tx complete
             reti               ; 15   ADC Conversion Complete
             reti               ; 16   EEPROM Ready
             reti               ; 17   Analog Comparator
             reti               ; 18   TWI (I²C) Serial Interface
             reti               ; 19   Store Program Memory Redy
;---------------------------------------------------------------
; Start, Power ON, Reset
main:        ldi    r16 , lo8(RAMEND)
             out    SPL , r16               ; Init Stackpointer LO
             ldi    r16 , hi8(RAMEND)
             out    SPH , r16
             ; Hier den Init-Code eintragen
;---------------------------------------------------------------
mainloop:    wdr
             ; Hier den Quellcode eintragen
             rjmp mainloop
;---------------------------------------------------------------
```

Quellcode in Assembler erstellen

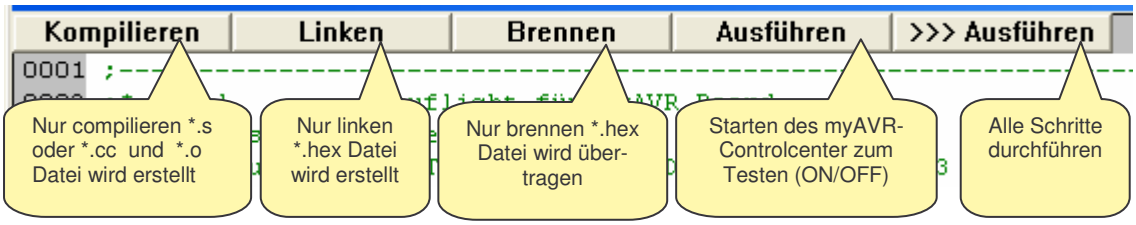
Das Lauflicht soll über die Ausgabegeräte LED angezeigt und von dem Prozessorport D gesteuert werden. Die Realisierung erfolgt über je ein Bit im Register R18. Dieses wird mit dem Befehl Bit-Rotation nach rechts verschoben und an PORT D des Prozessors ausgegeben. Auf Grund der Prozessorgeschwindigkeit muss die Ausgabe des Lauflichtes für unser Auge verzögert werden. Geben Sie folgenden Quellcode ein bzw. ergänzen Sie die Programmvorlage!

```
;-------------------------------------------------------------------------------
;* Titel            :Lauflicht für myAVR Board
;* Prozessor        :ATmega8 mit 3,6864 MHz
;* Schaltung        :PORT D.2 bis PORT D.4 an LED 1 bis 3
;* Datum            :15.03.2004
;* Autor            :Dipl. Ing. Päd. Alexander Huwaldt
;-------------------------------------------------------------------------------
.include            "avr.h"
;-------------------------------------------------------------------------------

; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:      rjmp    main        ; 1    POWER ON RESET
            reti                ; 2    Int0-Interrupt
            reti                ; 3    Int1-Interrupt
            reti                ; 4    TC2 Compare Match
            reti                ; 5    TC2 Overflow
            reti                ; 6    TC1 Capture
            reti                ; 7    TC1 Compare Match A
            reti                ; 8    TC1 Compare Match B
            reti                ; 9    TC1 Overflow
            reti                ; 10   TC0 Overflow
            reti                ; 11   SPI, STC Serial Transfer Complete
            reti                ; 12   UART Rx Complete
            reti                ; 13   UART Data Register Empty
            reti                ; 14   UART Tx complete
            reti                ; 15   ADC Conversion Complete
            reti                ; 16   EEPROM Ready
            reti                ; 17   Analog Comparator
            reti                ; 18   TWI (I²C) Serial Interface
            reti                ; 19   Store Program Memory Redy
;-------------------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16 , lo8(RAMEND)
            out     SPL , r16               ; Init Stackpointer LO
            ldi     r16 , hi8(RAMEND)
            out     SPH , r16               ; Init Stackpointer HI
            ldi     r16 , 0b11111111        ; PORT D auf Ausgang
            out     DDRD , r16
            ldi     r16 , 0b00000000        ; Alle Bits auf LOW
            out     PORTD , r16
            ldi     r17 , 0b00000000
            ldi     r18 , 0b00000001        ; 1 Lauflicht-Bit
;-------------------------------------------------------------------------------
mainloop:   wdr
            inc     r16                     ; Wait
            brne    skip
            inc     r17                     ; Wait
            brne    skip
            rcall   up1                     ; Lauflicht
skip:       rjmp    mainloop
;-------------------------------------------------------------------------------
up1:        rol     r18                     ; Bit-Rotation
            out     PORTD , r18
            ret
;-------------------------------------------------------------------------------
```

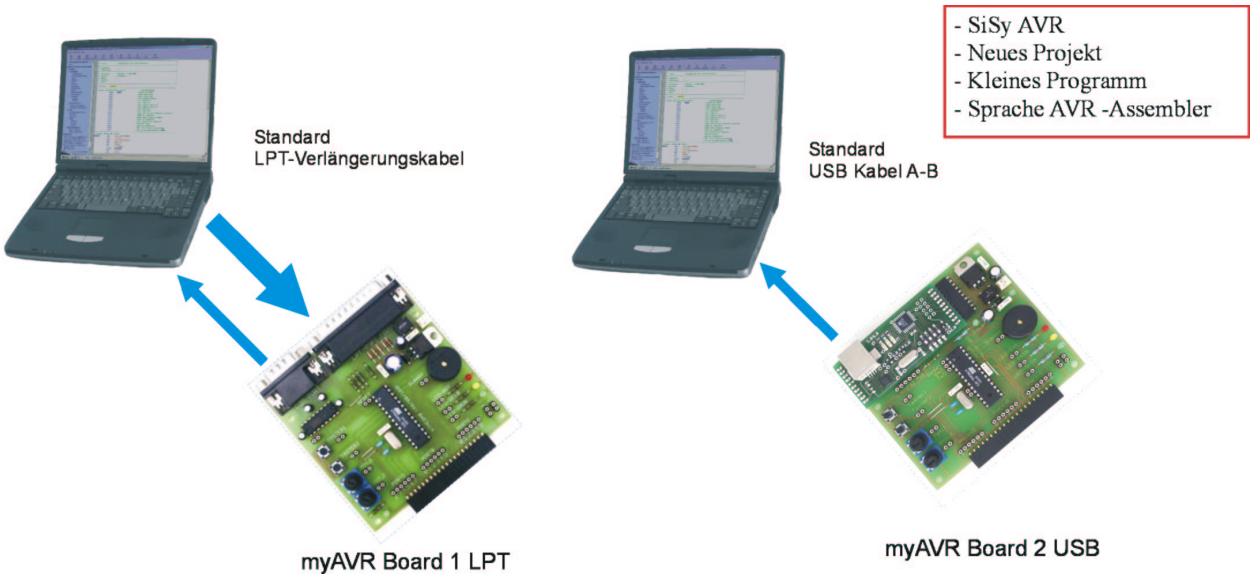
Compilieren und Linken

Der eingegebene Quellcode muss nun in Maschinencode für den AVR-Prozessor übersetzt werden. Wählen Sie dazu die Schaltflächen „Kompilieren“ und „Linken“. Bei fehlerfreier Übersetzung liegt das Programm als „Lauflicht.hex“ vor und kann auf den FLASH-Programmspeicher des Prozessors gebrannt werden.



Hardware anschließen und brennen

Das myAVR Board verfügt über eine ISP (In System Programming) Schnittstelle. Der Prozessor muss also nicht für die Programmierung aus dem System entfernt werden, um ihn in einem gesonderten Programmiergerät zu brennen, sondern kann im myAVR Board direkt programmiert werden. Dazu schließen Sie entsprechend Ihrer Boardvariante das Programmierkabel an den LPT- oder USB-Port Ihres Rechners an.



Zum Brennen wählen Sie die Schaltfläche „Brennen“. Bei erfolgreichem Brennvorgang erhalten Sie im Ausgabefenster folgende Meldung:

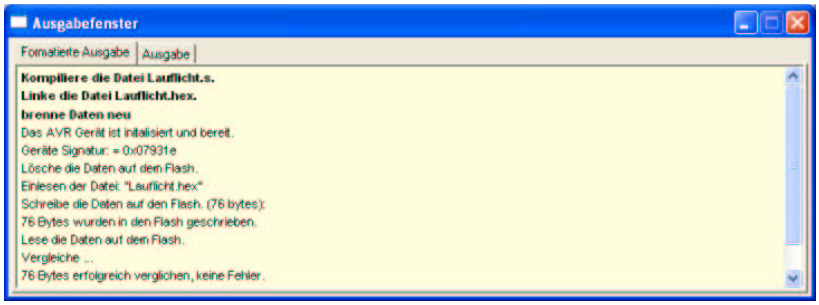

Ausgabefenster
Formatierte Ausgabe
Ausgabe
Kompiliere die Datei Lauflicht.s.
Linke die Datei Lauflicht.hex.
brenne Daten neu
Das AVR Gerät ist initialisiert und bereit.
Geräte Signatur: = 0x07931e
Lösche die Daten auf dem Flash.
Einlesen der Datei: "Lauflicht.hex"
Schreibe die Daten auf den Flash. (76 bytes):
76 Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Vergleiche ...
76 Bytes erfolgreich verglichen, keine Fehler.

Abbildung 7: Ausgabefenster

Mikrocontrollerlösung testen

Um das Programm zu testen ist es nötig, den Port D mit den Ausgabegeräten LED zu verbinden.

- Wenn vorhanden, ziehen Sie die Batterie bzw. das Netzteil und das Programmierkabel ab.
- Verbinden Sie die LEDs mit dem Prozessorport D entsprechend dem folgenden Schema.
  Nutzen Sie Patchkabel!
- Prüfen Sie die Verbindungen und schließen Sie die Batterie/das Netzteil oder das Programmierkabel wieder an und nehmen Sie die Mikrocontrollerlösung in Betrieb.
- Es ist jetzt an den LED's ein Lauflicht zu sehen.
- Gratulation!
  Das ist Ihre erste Mikrocontrollerlösung mit dem myAVR Board.

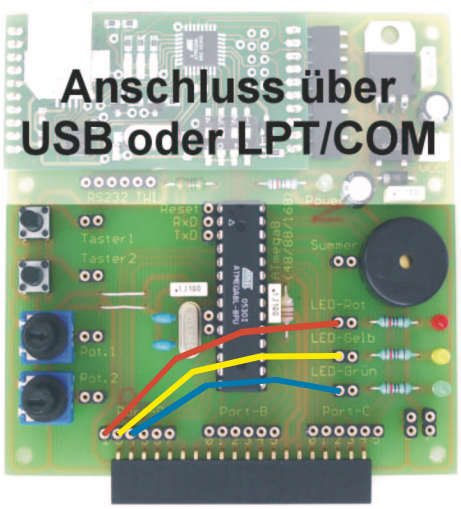


Abbildung 8: Verkablung

Beim Compilieren, Linken und Brennen des Schnellstart-Beispiels öffnete sich ein Ausgabefenster und zeigte Protokollausgaben der Aktionen an. Wenn die Hardware ordnungsgemäß angeschlossen, von der Software erkannt und das Programm erfolgreich auf den Programmspeicher des Mikrocontrollers übertragen wurde, muss die letzte Ausschrift folgenden Inhalt haben:

```
brenne Daten neu
Das AVR Gerät ist initalisiert und bereit.
Geräte Signatur: = 0x07931e
Lösche die Daten auf dem Flash.
Schreibe die Daten auf den Flash. (xxx bytes):
xxx Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Vergleiche ...
xxx Bytes erfolgreich verglichen, keine Fehler.
Ende.
```

Abbildung 9: Ausgabefenster mit "Brenn"-Protokoll

Inbetriebnahme, Test und Datenkommunikation mit der Mikrocontrollerlösung erfolgen über das myAVR Controlcenter. Dabei wird über die Schaltfläche „Start“ das Board mit der nötigen Betriebsspannung versorgt und der Controller gestartet. Der Datenaustausch mit dem myAVR Board ist möglich, wenn das Null-Modemkabel an Rechner und Board angeschlossen ist, sowie die Mikrocontrollerlösung dafür vorgesehen ist. Es können Texte und Bytes (vorzeichenlose ganzzahlige Werte bis 255) an das Board gesendet und Text empfangen werden. Die empfangenen Daten werden im Protokollfenster angezeigt. Vergleichen Sie dazu den folgenden Abschnitt zum myAVR-Controlcenter.

6 Das myAVR Controlcenter

6.1 Einleitung

Das myAVR Controlcenter ist ein universelles Terminalprogramm zur Kommunikation mit dem myAVR Board und anderen Mikrocontrollerapplikationen, die über eine serielle Schnittstelle (UART) oder USB Anbindung mit virtuellem COM-Port zum PC verfügen. Es kann für Test- und Debug-Meldungen sowie Visualisierung und Protokollierung von Messdaten genutzt werden. Dazu bietet das myAVR Controlcenter umfangreiche Konfigurationsmöglichkeiten.

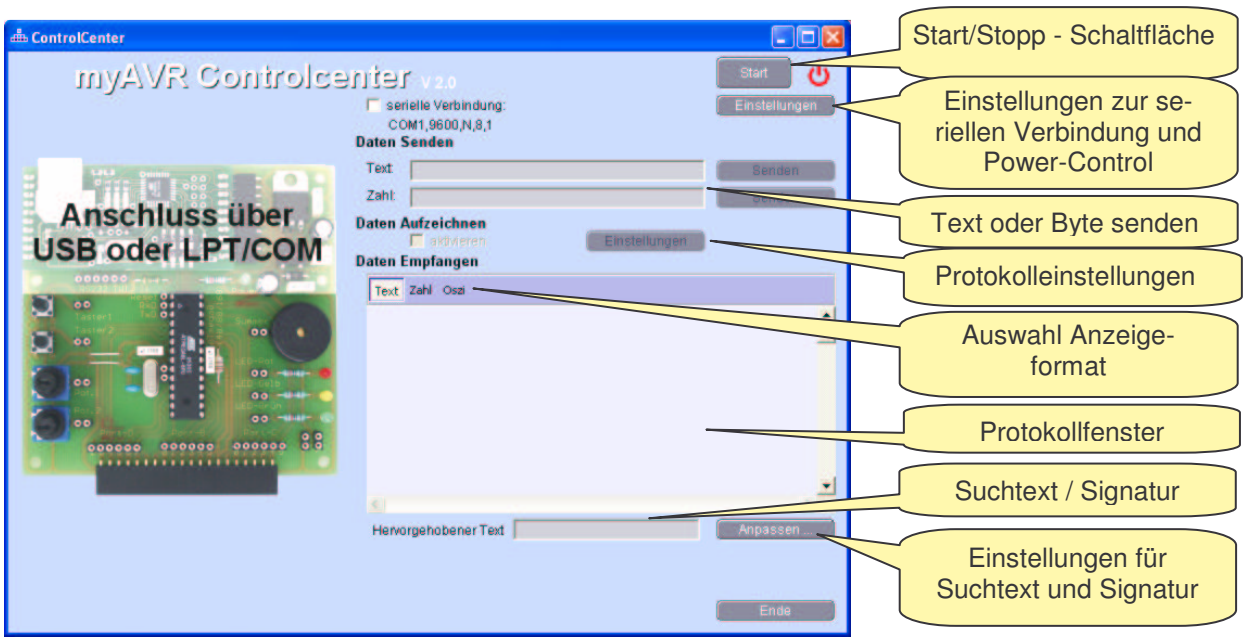
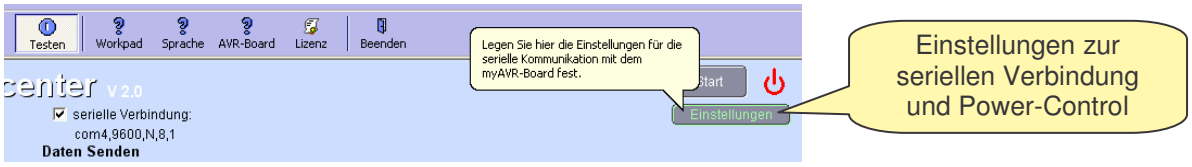
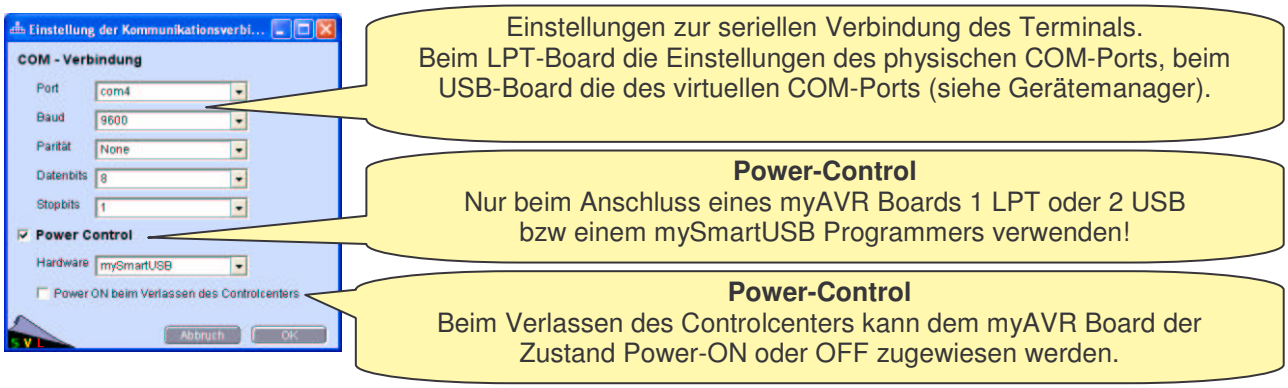


Abbildung 10: myAVR Controllcenter

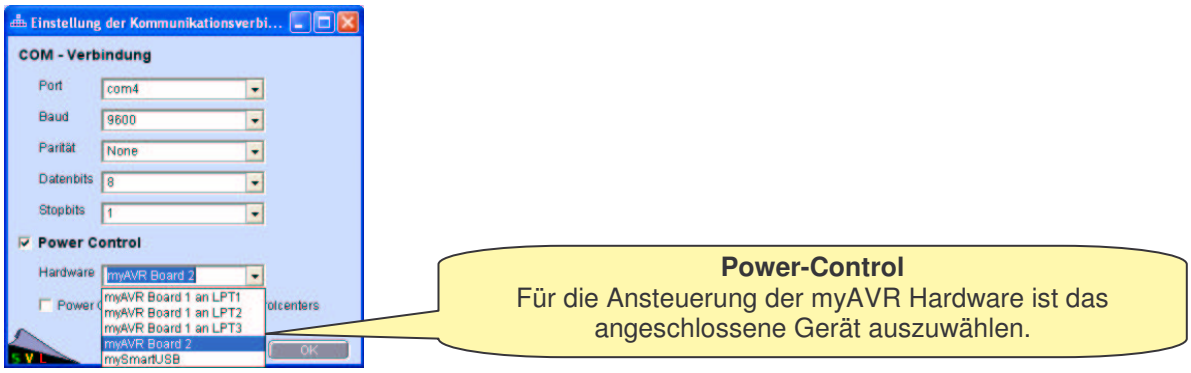
6.2 Das myAVR Board starten und stoppen (Power Control)

Wenn Sie ein myAVR Board Version 1 LPT oder Version 2 USB besitzen, können Sie die Spannungsversorgung des Boards aus dem Controlcenter heraus steuern. Dazu wählen Sie die Einstellungen für die serielle Verbindung und Power-Control.




Einstellung der Kommunikationsverbi...
COM - Verbindung
Port com4
Baud 9600
Parität None
Datenbits 8
Stopbits 1
Power Control
Hardware mySmartUSB
Power ON beim Verlassen des Controlcenters
Abbruch
OK
Einstellungen zur seriellen Verbindung des Terminals.
Beim LPT-Board die Einstellungen des physischen COM-Ports, beim USB-Board die des virtuellen COM-Ports (siehe Gerätemanager).
Power-Control
Nur beim Anschluss eines myAVR Boards 1 LPT oder 2 USB bzw einem mySmartUSB Programmers verwenden!
Power-Control
Beim Verlassen des Controlcenters kann dem myAVR Board der Zustand Power-ON oder OFF zugewiesen werden.

Mit dem Aktivieren der Power-Control-Funktion wird beim Betätigen der Schaltfläche START/STOPP die rechnerseitige Spannungsversorgung des myAVR Boards und die RESET-Leitung angesteuert.

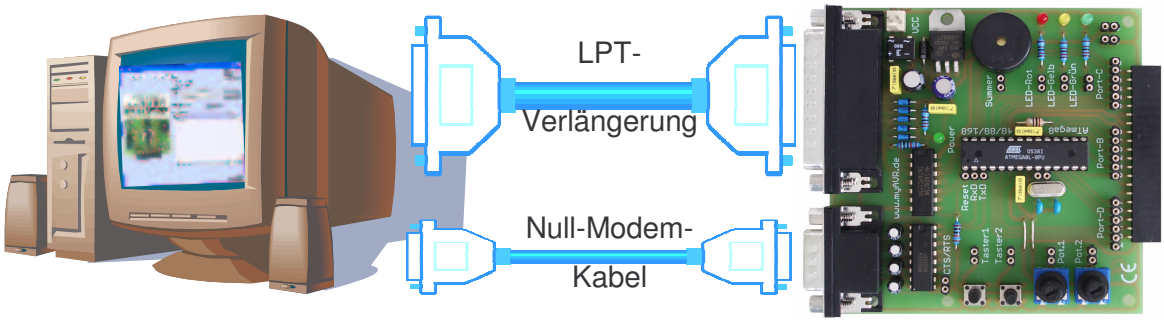


Bei einer externen Spannungsversorgung, zum Beispiel durch einen 9V Block oder das myAVR Netzteil, hat die Power-Control-Funktion keine Wirkung auf die Boardspannung. In diesem Fall bezieht sich die Wirkung ausschließlich auf die RESET-Leitung, um den Controller zu starten bzw. zu stoppen. Beachten Sie, dass beim myAVR Board 1 LPT die Spannungsversorgung über den LPT-Port für eine serielle Kommunikation und den Betrieb des LC-Displays nicht ausreicht, dafür wird die Verwendung eines myAVR Netzteils empfohlen.

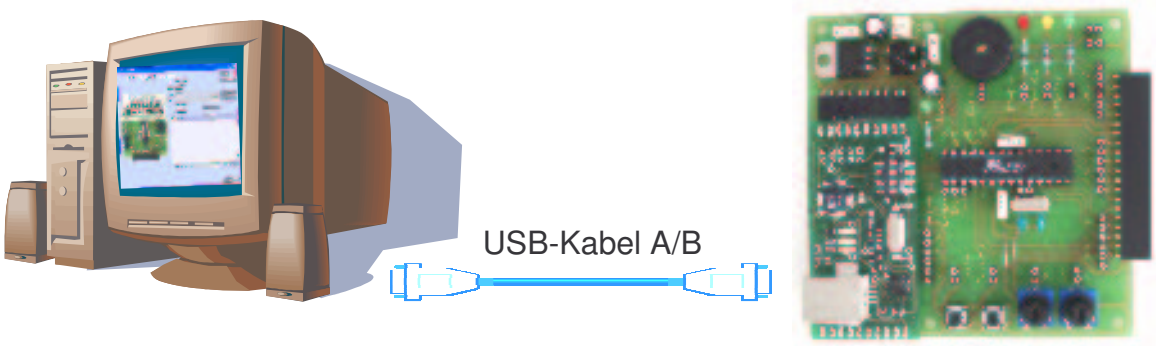
6.3 Kommunikation mit dem myAVR Board

6.3.1 Grundlagen (LPT/USB-Variante)

Die zwei myAVR Boardversionen unterscheiden sich grundsätzlich in der technischen Realisierung der seriellen Kommunikation. Das myAVR Board 1 LPT verfügt über eine gesonderte RS232 Schnittstelle (COM-Port) und wird über ein Nullmodemkabel an einen physisch vorhandenen COM-Anschluss des PC angeschlossen.

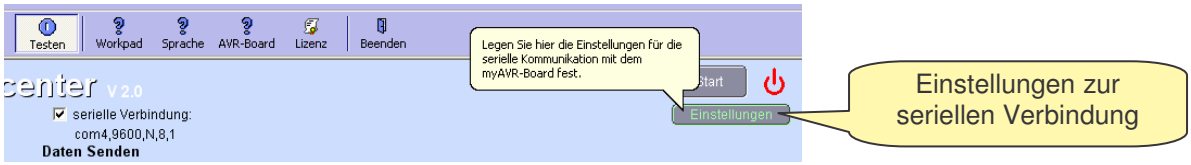


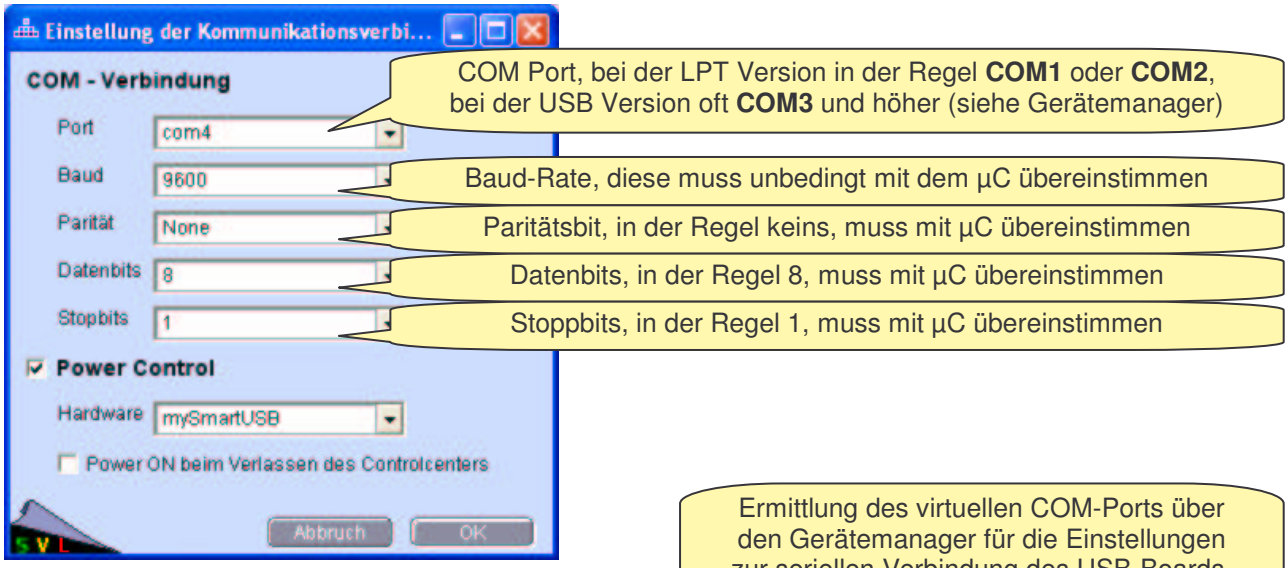
Das myAVR Board 2 USB verfügt über den USB Programmer mySmartUSB. Dieser stellt gleichzeitig einen virtuellen COM-Port für die Kommunikation zur Verfügung.

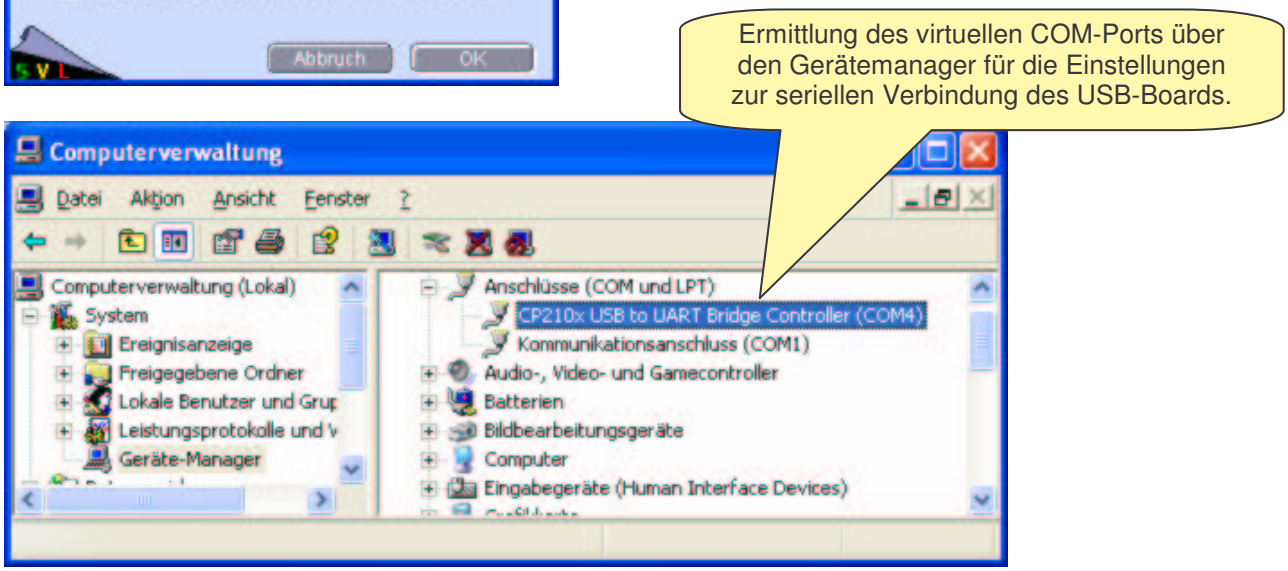


6.3.2 Einstellungen für die seriellen Verbindung

Für eine erfolgreiche Kommunikation mit dem myAVR Board ist es wichtig, dass Sender und Empfänger von seriellen Daten die gleichen Parameter für die Datenübertragung konfiguriert haben. Auf der PC Seite werden die Kommunikationsparameter im Controlcenter über die Schaltfläche für die Einstellungen der Verbindung konfiguriert.







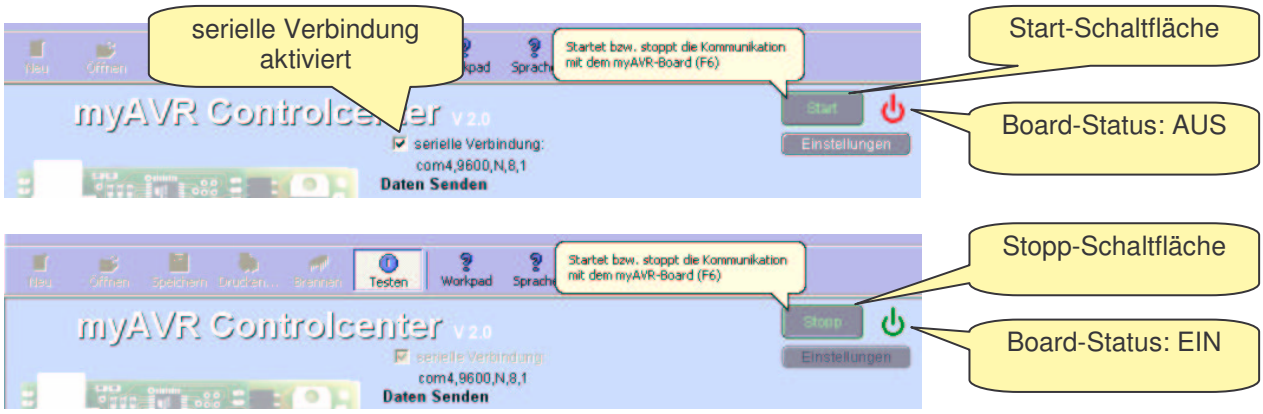
Im Mikrocontrollerprogramm sind die gleichen Parameter bei der Initialisierung der UART zu wählen. Beachten Sie die eingestellte Taktquelle (hier Quarz mit 3,6 MHz).

```
#define     F_CPU   3686400
#define     BAUD    9600
#include    <avr\io.h>
//-------------------------------------------------------------------
// UART initialisieren
void initUART()
{
    sbi(UCSRB,3);   // TX aktiv
    sbi(UCSRB,4);   // RX aktivieren
    UBRRL=(uint8_t)(F_CPU/(BAUD*16L))-1;        // Baudrate festlegen
    UBRRH=(uint8_t)((F_CPU/(BAUD*16L))-1)>>8;   // Baudrate festlegen
}
```

Beispiel für die Konfiguration der UART des Mikrocontrollers (µC) mit 9600 Baud.

6.3.3 Daten empfangen vom myAVR Borad

Das myAVR Controlcenter empfängt Daten über den gewählten COM-Port und stellt diese im Protokollfenster dar. Damit können Statusmeldungen, Fehlermeldungen oder auch Messwerte erfasst werden. Die Voraussetzung ist, dass die serielle Verbindung hergestellt (Nullmodemkabel oder USB Kabel), korrekt konfiguriert und aktiviert wurde. Die Kommunikation beginnt mit dem Betätigen der Schaltfläche „Start“ und endet mit dem Betätigen der Schaltfläche „Stopp“. Der Zustand (in Betrieb/Halt) wird über das Symbol (rot/grün, EIN/AUS) rechts neben der Schaltfläche angezeigt.



6.3.4 Darstellung der empfangen Daten

Die empfangenen Daten werden fortlaufend im Protokollfenster dargestellt. Der Darstellungsmodus kann während der Kommunikation umgeschaltet werden. Das Controlcenter bietet folgende Darstellungsmodi:

- Text
- Zahl
  - Dezimal
  - Hexadezimal
- Grafik (Oszi)

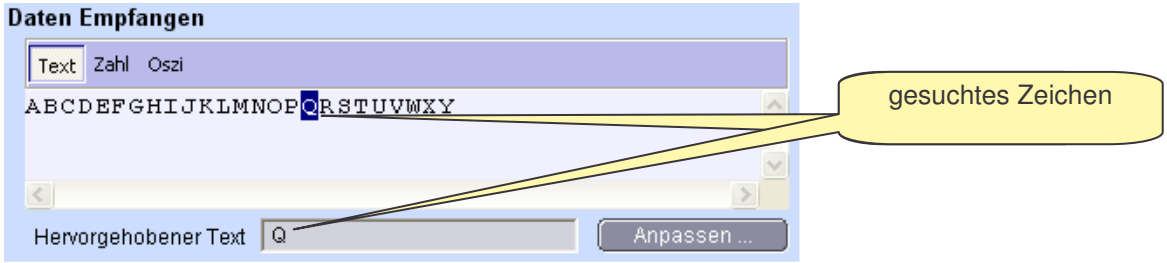
Die Daten des folgenden Programmbeispiels sollen als Testdaten dienen:

```
//------------------------------------------------------
main()
{
    uint8_t a='A';              // Zeichen
    initUART();                 // Initialisierungen
    while (true)                // Mainloop-Begin
    {
        putChar(a);             // Zeichen senden
        a++;                    // nächstes Zeichen
        myWait_1000ms();        // Pause
    }                           // Mainloop-Ende
}
//------------------------------------------------------
```

Der Textmodus

Der Textmodus dient zur Visualisierung alphanumerischer Werte im ASCII Format (Zeichenketten). Zahlen, die mit Zeichenketten gesendet werden, müssen vom Mikrocontrollerprogramm zuvor ins ASCII-Format gewandelt werden (siehe `itoa` und `sprintf`).

Über die Schaltfläche „Anpassen “ lässt sich das Protokollfenster weiter konfigurieren. Die Zeilenbreite und das Zeichen für einen Zeilenumbruch lassen sich auswählen. Es können wichtige/gesuchte Textstellen im Protokollfenster hervorgehoben werden.

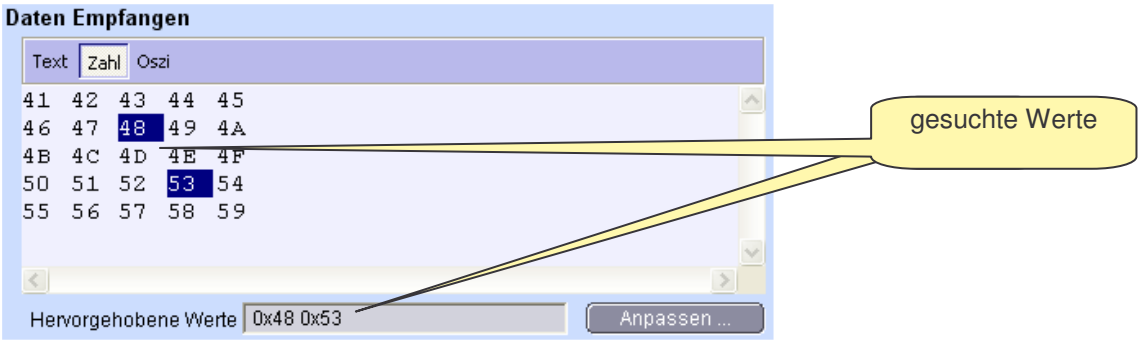


Der Zahlenmodus

Der Zahlenmodus visualisiert die empfangenen Datenbytes (8 Bit, Werte von 0 bis 255) als Dezimal- oder Hexadezimalzahlen. Dezimalzahlen werden dreistellig mit führender Null dargestellt.

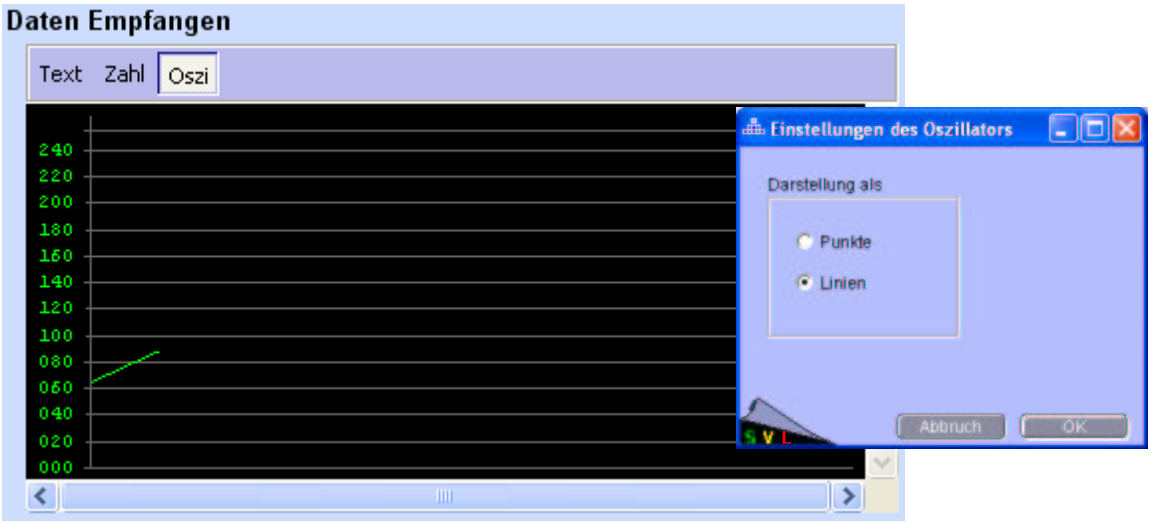
Die Anzeige kann über die Schaltfläche „Anpassen“ weiter konfiguriert werden. Dabei werden die Zeilengröße oder das Zeilenumbruchzeichen sowie das Format der Zahlendarstellung (Hexadezimal/Dezimal) ausgewählt.

Es können wichtige/gesuchte Zahlenwerte im Protokollfenster hervorgehoben werden. Hezadezimale Zahlen sind durch den Präfix 0x zu kennzeichnen. Mehrere Werte können mit Leerzeichen getrennt angegeben werden.



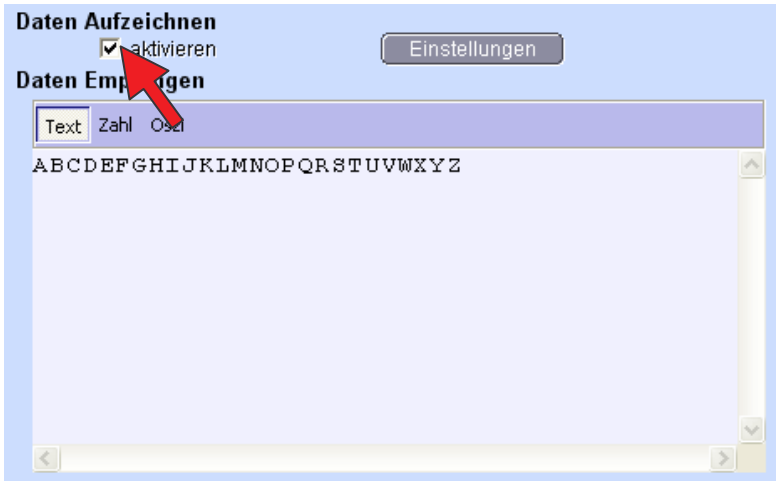
Der Grafikmodus

Messwerte können auch grafisch visualisiert werden. Dabei werden die Werte fortlaufend und byteweise (Wertebereich 0 bis 255) als Punkte in einem Koordinatensystem visualisiert. Die X-Achse repräsentiert den zeitlichen Verlauf, die Y-Achse den Wertebereich der Daten. Die Anzeige kann als einzelne Punkte oder Linien erfolgen.

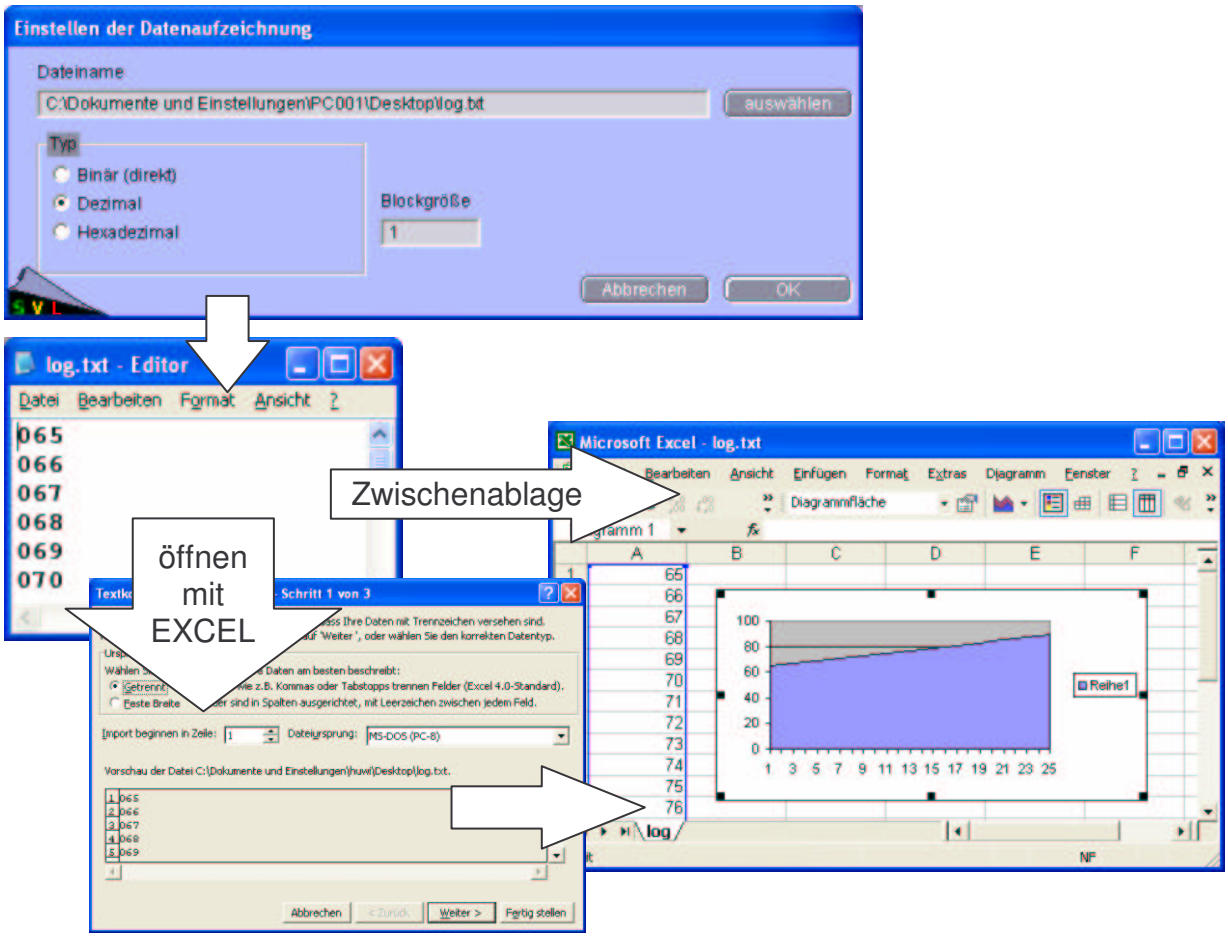


6.3.5 Empfangene Daten speichern

Bei der Erfassung von Messdaten die vom Mikrocontroller an den PC gesendet werden ist es oft wichtig, diese in eine Datei zu speichern. Damit wird eine Weiterverarbeitung der Daten in entsprechenden Programmen (z.B.: EXCEL) möglich. Das myAVR Controlcenter ermöglicht es, Protokolldaten aufzuzeichnen (Log-Datei, Rekorderfunktion). Um diese Funktion zu aktivieren, wählen Sie die Option „Daten aufzeichnen“



Über die Schaltfläche „Einstellungen“ muss der Dateiname und Pfad der Log-Datei angegeben werden. Zusätzlich ist es möglich, das Format und die Blockgröße (Zeilenumbruch) festzulegen. Die Weiterverarbeitung der Daten erfolgt entsprechend der Möglichkeiten der Zielanwendung (z.B.: Öffnen, Importieren oder Zwischenablage).



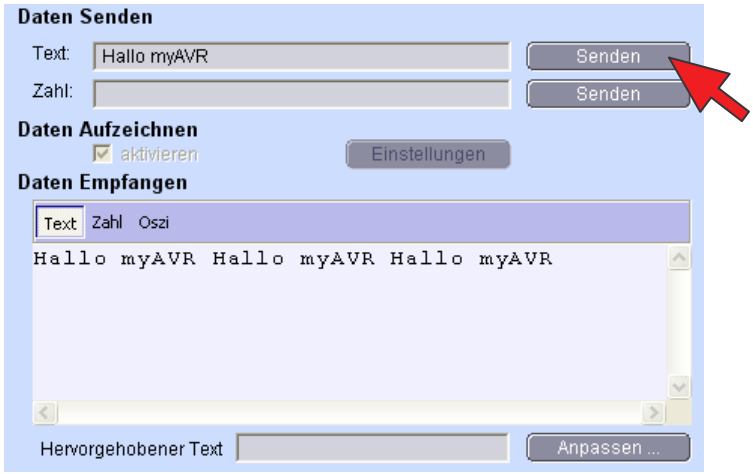
6.3.6 Daten an das myAVR Board senden

Über das Controlcenter können Daten im Text- oder Zahlenformat an den Mikrocontroller gesendet werden. Das folgende Programm dient als Veranschaulichung dieser Funktion.

```
//------------------------------------------------------
main()
{
    uint8_t zeichen;
    initUART();                 // Initialisierungen
    while (true)                // Mainloop-Begin
    {
        zeichen=getChar();      // Zeichen abholen
        putChar(zeichen);       // Zeichen zurücksenden
    }                           // Mainloop-Ende
}
//------------------------------------------------------
```

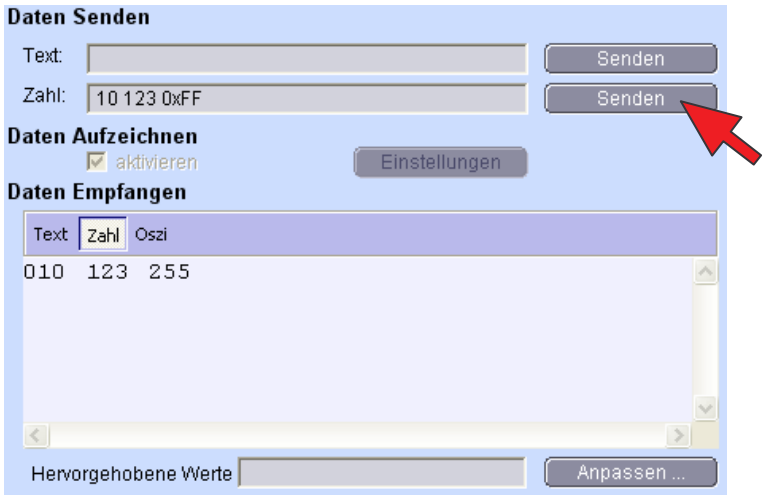
Text senden

Es können einzelne Zeichen, aber auch Zeichenketten gesendet werden. Mit der zugehörigen Schaltfläche „Senden“ wird immer der gesamte Inhalt der Eingabezeile „Text“ gesendet.



Zahlen senden

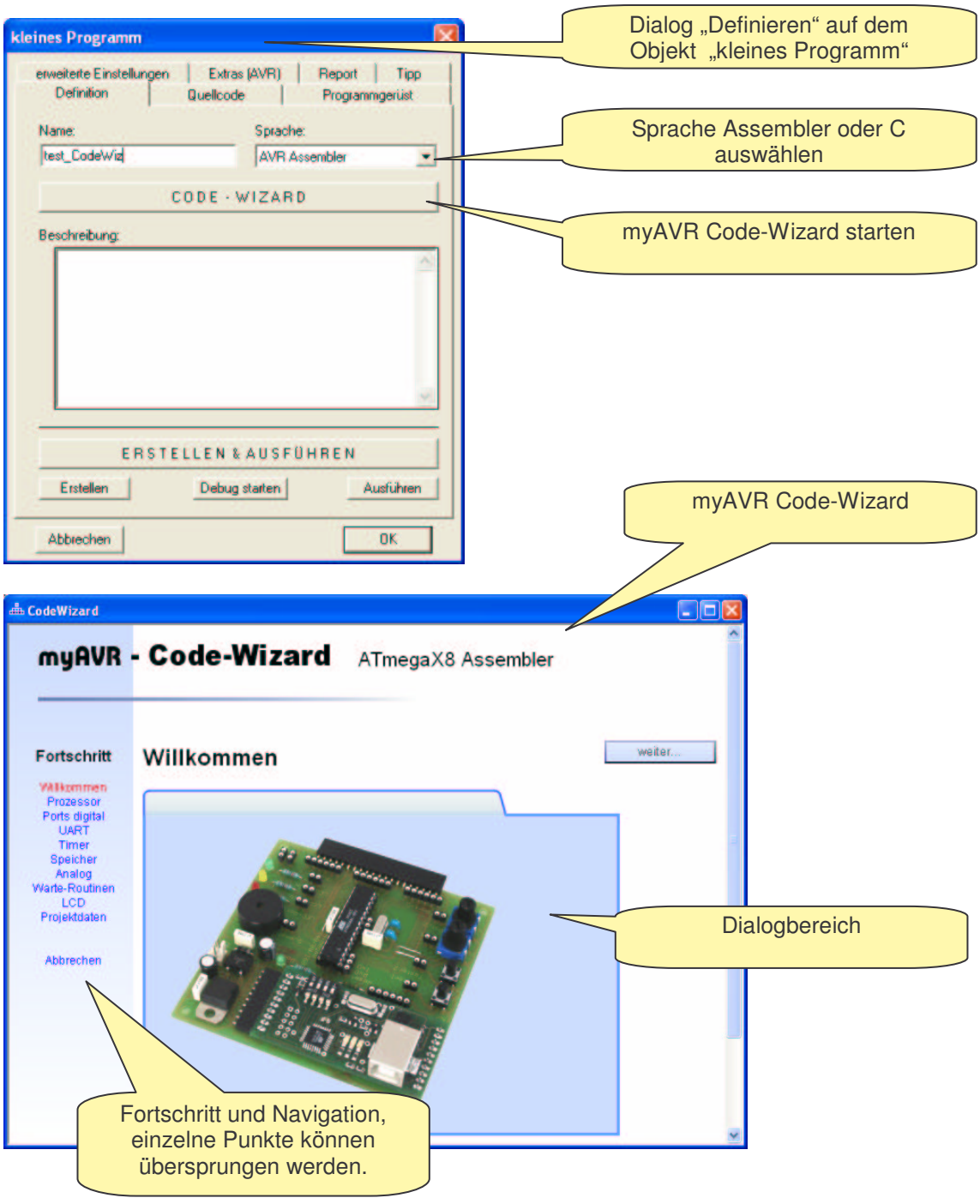
Zahlenwerte von 0 bis 255 (1 Byte) können einzeln oder als Zahlenfolge gesendet werden. Zwischen den Werten ist ein Leerzeichen als Trennzeichen einzufügen. Zahlen, die größer sind als 255, werden auf den niederwertigen Teil gekürzt. Es ist möglich, Zahlen im Hexadezimalformat zu schreiben (Präfix 0x). Mit der zugehörigen Schaltfläche „Senden“ wird immer der gesamte Inhalt der Eingabezeile „Zahl“ gesendet.



7 Der myAVR Code-Wizard

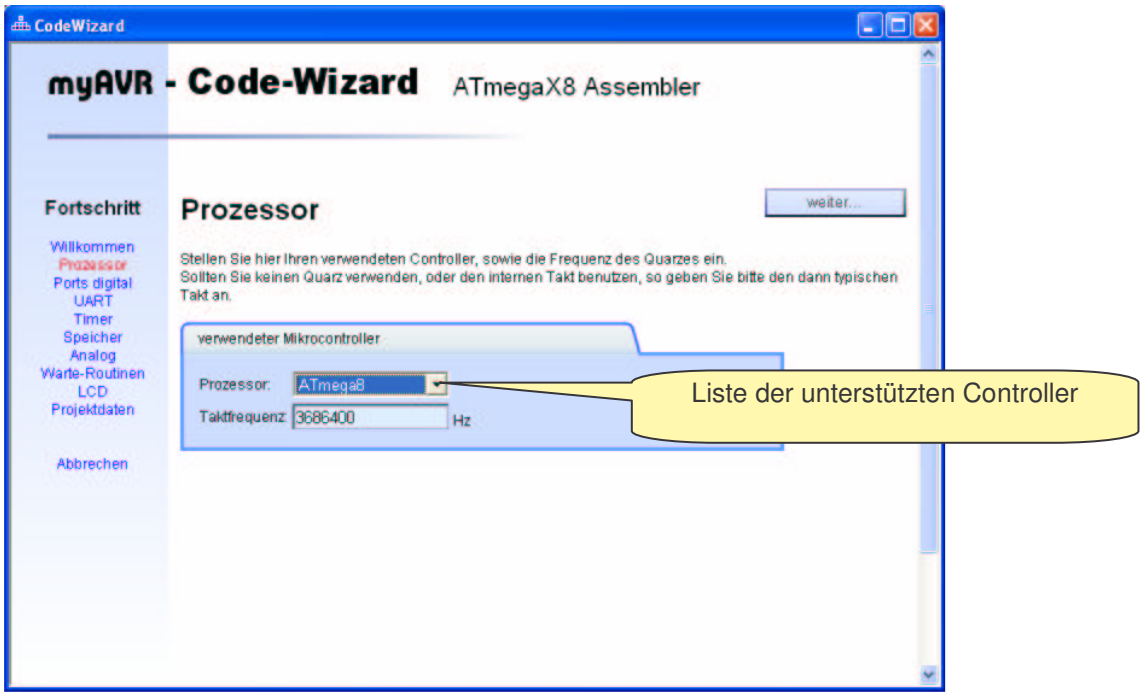
7.1 Einführung

Der myAVR Code-Wizard ist ein Assistent zum Erstellen von Assembler- und C-Codes für die Konfiguration und Anwendungsentwicklung von AVR Mikrocontrollern. Dabei wählt der Nutzer Schritt für Schritt im Dialog mit dem Assistenten die gewünschten Konfigurationen und Programmbausteine aus. Der Code-Wizard generiert kompilierungsfähigen Quellcode in der gewünschten Programmiersprache (AVR C oder AVR Assembler), der als komplette Anwendung geladen wird. Der Entwickler muss nur noch die projektspezifische individuelle Logik ergänzen. Der gesamte Programmrahmen in Form von Hauptprogramm, fertige Initialisierungssequenz, Unterprogramme und Interruptroutinen wird von Code-Wizard generiert.



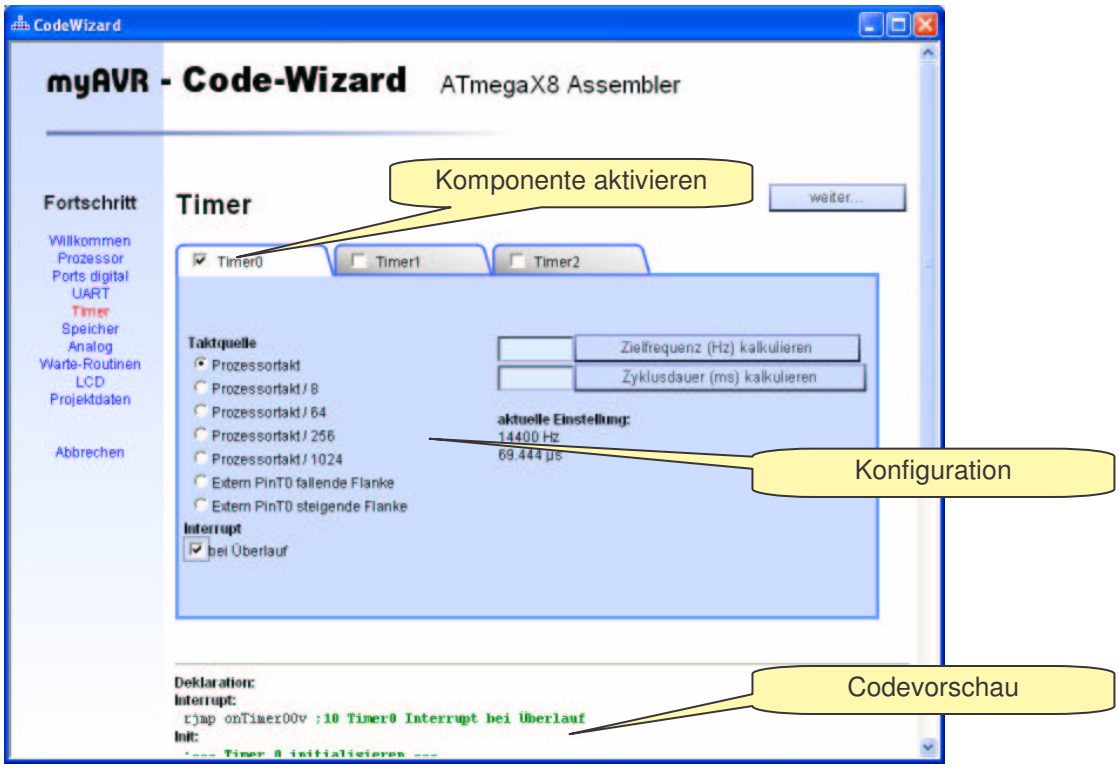
7.2 Grundeinstellungen

Die Zielsprache wurde vor dem Start des Code-Wizards ausgewählt. Für die Generierung von korrektem Quellcode ist es notwendig, den Controllertyp und die Taktgeschwindigkeit festzulegen.



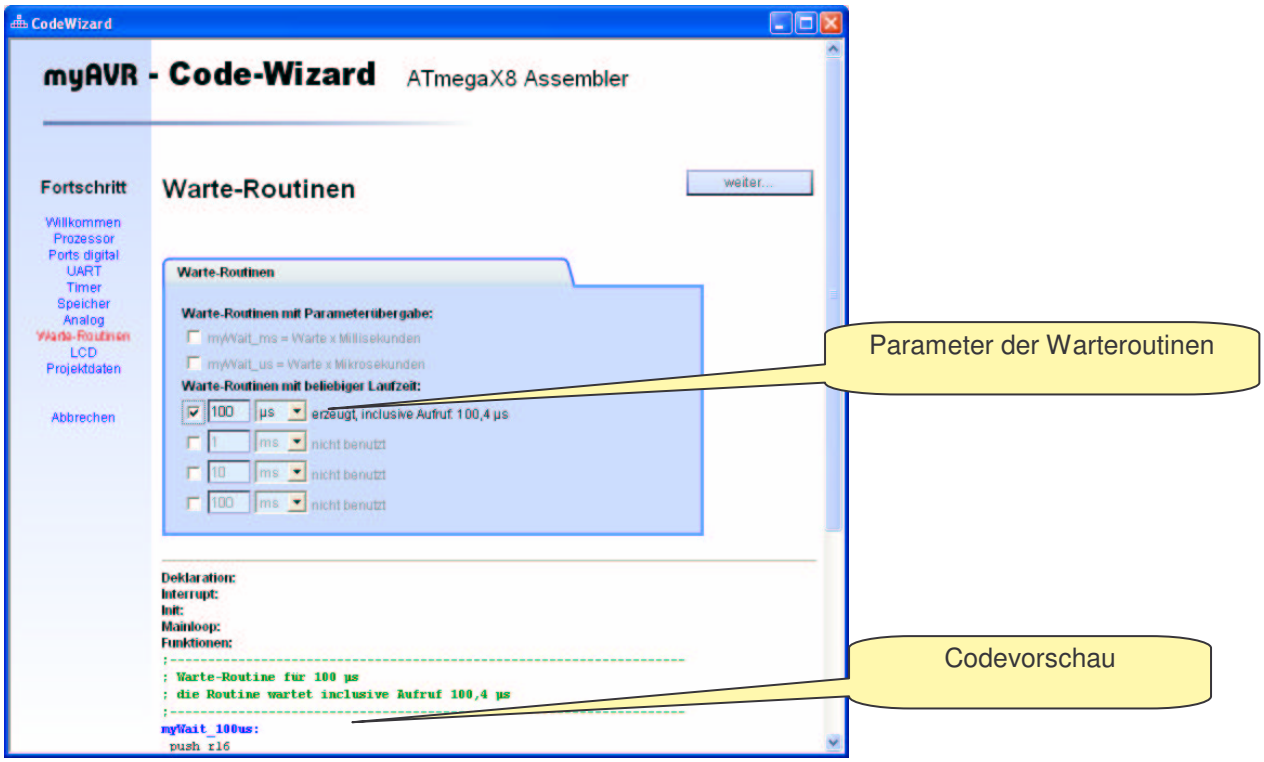
7.3 Geräteeinstellungen

Über den Dialogbereich können Schritt für Schritt die Komponenten des gewählten Controllertyps konfiguriert werden.



7.4 Unterprogramme

Neben der Möglichkeit die Hardwarekomponenten des gewünschten Controllers zu konfigurieren, bietet der Code-Wizard auch eine Reihe typischer Unterprogramme. So verfügt er über einen Warteroutinen-Rechner zum generieren präziser Wartefunktionen.



7.5 Projektdaten

Der letzte Punkt im Code-Wizard ist die Eingabe der Projektdaten. Aus diesen wird eine Programmkopfdokumentation generiert und dem kompletten Quellcode vorangestellt.

7.6 Codegenerierung

Der vollständige Quellcode wird zur Kontrolle dem Anwender angezeigt. Es können jetzt noch Änderungen vorgenommen werden, indem man die betreffenden Punkte im Navigationsbereich auswählt und die Parameter ändert. Mit Bestätigung des Quellcodes wird dieser als komplettes Programm eingefügt.

```
//---------------------------------------------------------------------------
// Titel : Test der seriellen Verbindung
//---------------------------------------------------------------------------
// Funktion : empfängt und sendet Daten per UART (9600,8,n,1)
// Schaltung : Nullmodemkabel anschließen
//---------------------------------------------------------------------------
// Prozessor : ATmega8
// Takt : 3686400 Hz
// Sprache : C
// Datum : 9.7.2006
// Version : 1.0
// Autor : Dipl. Ing. Päd. Huwaldt
// Programmer: SP12
// Port : LPT1
//---------------------------------------------------------------------------
// created by myAVR-Code-Wizard
//---------------------------------------------------------------------------
#define   F_CPU   3686400
#include <avr\io.h>
#include <avr\interrupt.h>
//---------------------------------------------------------------------------
// Initialisierungen
//---------------------------------------------------------------------------
void init()
{
    // Ports initialisieren
    ...
```

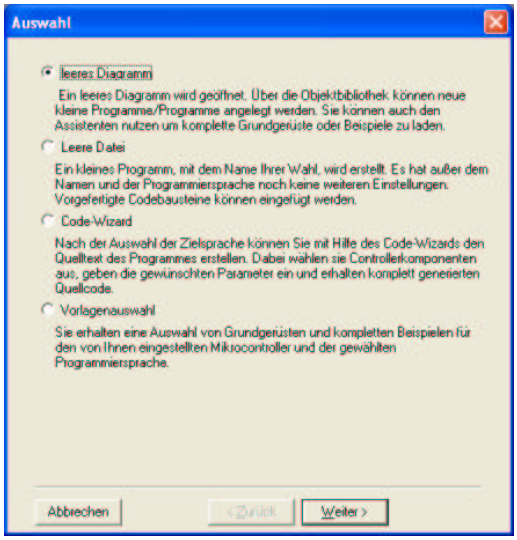
8 Entwicklung eines großen Programms mit SiSy AVR

8.1 Einleitung

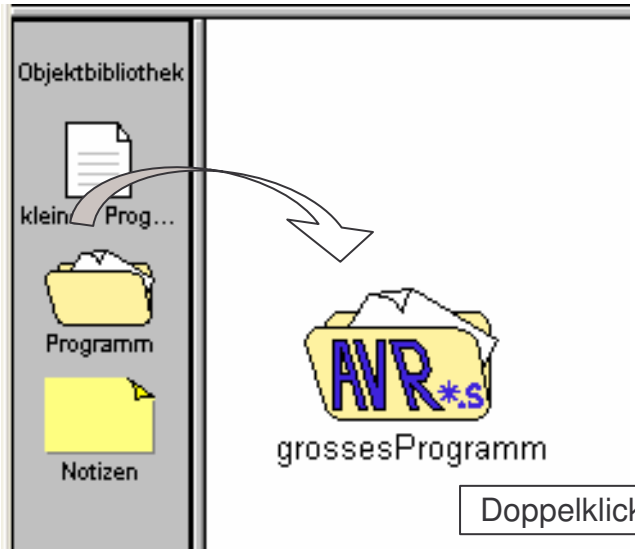
Für die Entwicklung eines größeren Programms ist es unzweckmäßig, alle Befehle in eine Datei (Unit) zu schreiben. Der Grund dafür ist, dass bei mehr als 60 bis 80 Zeilen Quellcode die Übersicht über die Programmstruktur verloren geht. Selbst die Unterteilung in Unterprogramme reicht ab einer bestimmten Größe von Programmen nicht mehr aus. SiSy erlaubt zwar in kleinen Programmen bzw. je Unit 10 Kilobyte Code. Das sind in Assembler zum Beispiel über 1000 Zeilen. Ein kleines Programm bzw. eine einzelne Unit sollte jedoch nie mehr als 80 bis 120 Zeilen haben. Wird diese Grenze erreicht, sollte das Programm in logische Einheiten (Units, Module) gegliedert werden. Dabei fasst man alles zusammen, was zu einer bestimmten Funktionalität oder einer bestimmten Baugruppe wie zum Beispiel dem AD-Wandler gehört. Physisch entstehen dabei mehrere Dateien, die für sich genommen wieder übersichtlich sind da diese dann nur 80 bis 120 Zeilen Code enthalten. Das Übersetzungsprogramm (Assembler, Compiler, Linker) sorgt dann dafür, dass alle einzelnen Units zu einem vollständigen Programm zusammengefügt werden.
Das folgende Kapitel erläutert an einem sehr kleinen Beispiel die Handhabung der Komponenten eines großen Programms, welches in mehrere Units zerlegt wird.

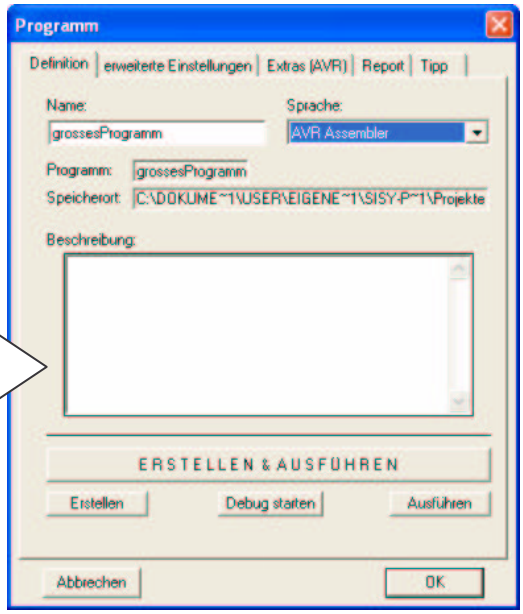
8.2 Vorbereitung

Starten Sie SiSy und legen Sie ein neues Projekt an. Wählen Sie das Vorgehensmodell „Programmierung“. Nehmen Sie die Grundeinstellungen für die verwendete AVR Hardware vor oder lassen Sie die myAVR-Hardware automatisch suchen. Erstellen Sie ein leeres Diagramm!


Auswahl
leeres Diagramm
Ein leeres Diagramm wird geöffnet. Über die Objektbibliothek können neue kleine Programme/Programme angelegt werden. Sie können auch den Assistenten nutzen um komplette Grundgerüste oder Beispiele zu laden.
Leere Datei
Ein kleines Programm, mit dem Name Ihrer Wahl, wird erstellt. Es hat außer dem Namen und der Programmiersprache noch keine weiteren Einstellungen. Vorgefertigte Codebausteine können eingefügt werden.
Code-Wizard
Nach der Auswahl der Zielsprache können Sie mit Hilfe des Code-Wizards den Quelltext des Programmes erstellen. Dabei wählen sie Controllerkomponenten aus, geben die gewünschten Parameter ein und erhalten komplett generierten Quellcode.
Vorlagenauswahl
Sie erhalten eine Auswahl von Grundgerüsten und kompletten Beispielen für den von Ihnen eingestellten Mikrocontroller und der gewählten Programmiersprache.
Abbrechen
< Zurück
Weiter >

Ziehen Sie aus der Objektbibliothek ein Objekt vom Typ „Programm“.





Legen Sie Name und Sprache für das Programm fest. Überprüfen Sie gegebenenfalls die Einstellungen unter Extras AVR.

8.3 Aufgabenstellung

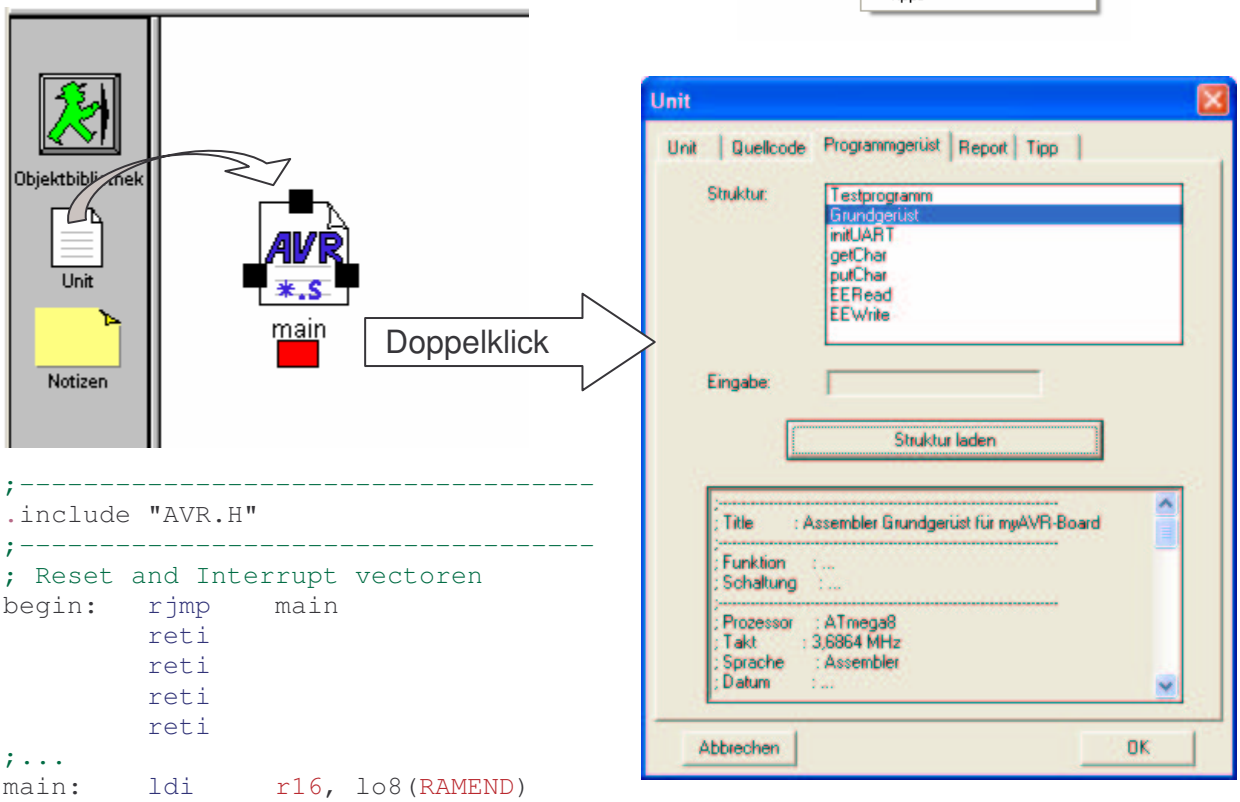
Es ist eine Mikrocontroller-Anwendung mit der Technik des großen Programms (Zerlegen in mehrere Units) zu entwerfen und in der Sprache Assembler zu realisieren.

Aufgabe:
Entwickeln Sie eine Mikrocontrollerlösung, bei der ein Taster eine LED schaltet.

Schaltung:
Port B.0 = Taster 1
Port B.1 = LED

8.4 Hauptprogramm erstellen

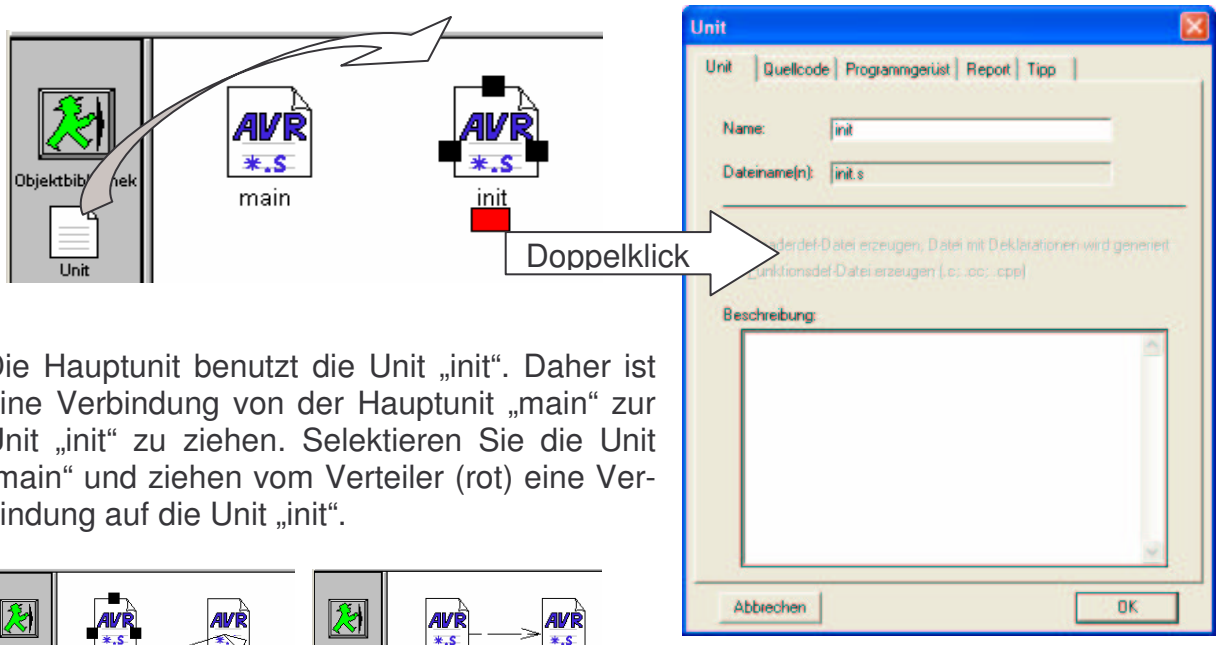
Öffnen Sie das Diagrammfenster für ein großes Programm, indem Sie auf dem Symbol rechte Maustaste -> Kontextmenü -> nach unten (öffnen) wählen. Legen Sie eine Unit an. Diese Unit bildet das Hauptprogramm. Nennen Sie die Unit „main“ und wählen die Sprache Assembler. Damit wird durch die Entwicklungsumgebung erkannt, dass es sich hierbei um das Hauptmodul handelt. Erstellen Sie hier das Hauptprogramm, nutzen Sie die angebotene Vorlage „Grundgerüst“.



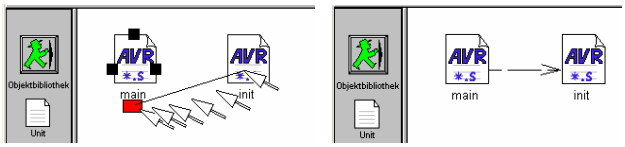

```
;-----------------------------------
.include "AVR.H"
;-----------------------------------
; Reset and Interrupt vectoren
begin:    rjmp    main
          reti
          reti
          reti
          reti
;...
main:     ldi     r16, lo8(RAMEND)
          out     SPL, r16
          ldi     r16, hi8(RAMEND)
          out     SPH, r16
          ; Hier Init-Code eintragen.
;-----------------------------------
mainloop: wdr
          ; Hier Quellcode eintragen.
          rjmp mainloop
;-----------------------------------
```

8.5 Units (Unterprogramme) anlegen und verknüpfen

Zur Gliederung des großen Programms wird dieses in mehrere kleine Einheiten (Units/ Module) zerlegt. Diese Einheiten werden nach fachlichen/inhaltlichen Gesichtspunkten gebildet. So kann man alle Initialisierungsaufgaben in der Unit „init“ zusammenfassen. Eine Unit kann aus einer Funktion/Unterprogramm oder mehreren Funktionen/Unterprogrammen bestehen. Im einfachsten Fall enthält jede Unit ein Unterprogramm. Legen Sie zusätzlich zur Haupt-Unit „main“ die Unit „init“ an.

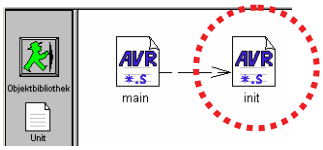


Die Hauptunit benutzt die Unit „init“. Daher ist eine Verbindung von der Hauptunit „main“ zur Unit „init“ zu ziehen. Selektieren Sie die Unit „main“ und ziehen vom Verteiler (rot) eine Verbindung auf die Unit „init“.



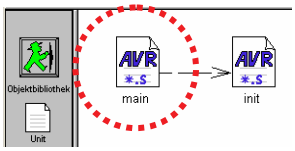
Dabei wird in der Hauptunit „main“ ein Include-Eintrag für die Unit „init“ erzeugt. Erstellen Sie die Initialisierungsroutine für die benötigten digitalen Ein und Ausgänge.

```
;---------------------------------------
init:       push    r16
            sbi     DDRB,0    ; B.0 Ausgang
            cbi     PORTB,0   ; B.0 LED aus
            cbi     DDRD,2    ; D.2 Eingang
            sbi     PORTD,2   ; D.2 PullUp
            pop     r16
            ret
;---------------------------------------
```



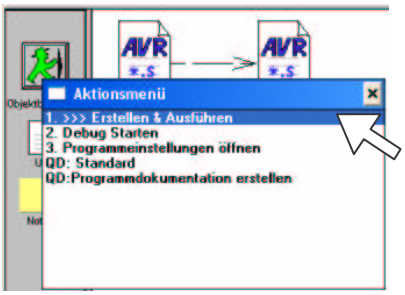
Ergänzen Sie den Code des Hauptprogramms.

```
            ...
            rcall   init
;-----------------------------------
mainloop:   sbic    PIND,2
            rjmp    led_aus
led_an:     sbi     PORTB,0
            rjmp    mainloop
led_aus:    cbi     PORTB,0
            rjmp    mainloop
;-----------------------------------
.include "init.s"
```



8.6 Übersetzen, Brennen und Test

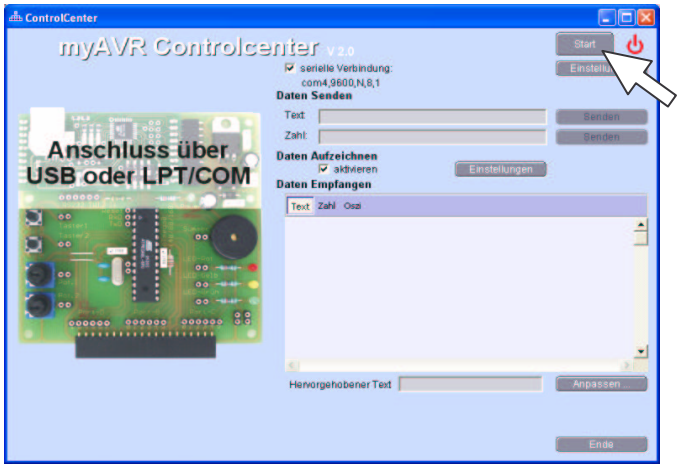
Zum Übersetzen, Brennen und Testen wählen Sie im Aktionsmenü (Ampelmännchen) den entsprechenden Menüpunkt. Im Ausgabefenster erscheint das Protokoll der ausgeführten Aktionen.



Kompiliere die Datei main.s.
Linke die Datei grossesProgramm.hex.

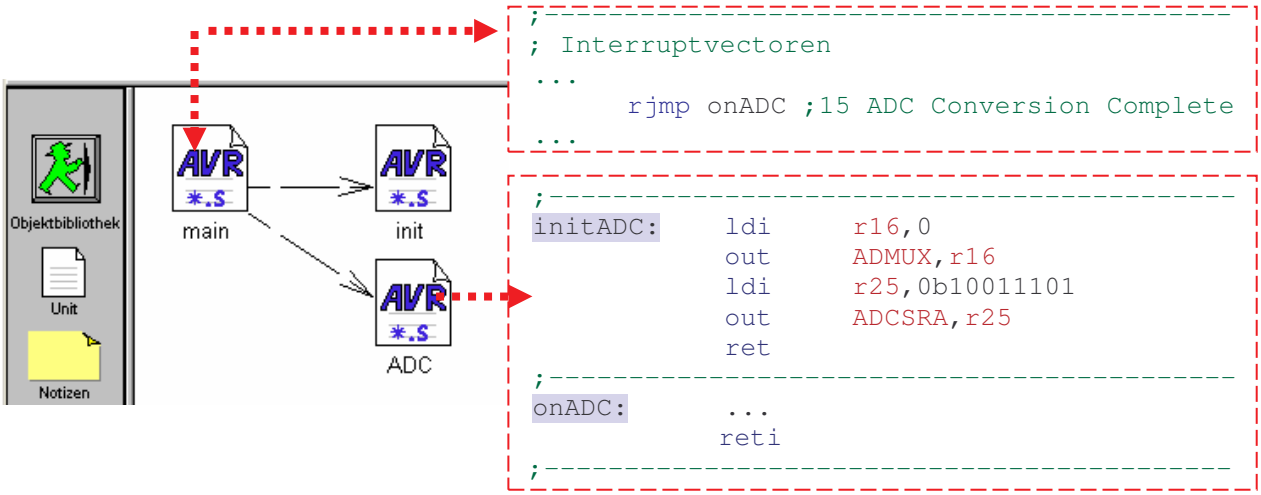
brenne Daten neu
Das AVR Gerät ist initalisiert und bereit.
…
74 Bytes wurden in den Flash geschrieben.
…
74 Bytes erfolgreich verglichen, keine Fehler.
Starte Debug...

Über das myAVR Controlcenter können Sie das myAVR Board Starten und Stoppen um die Mikrocontrollerlösung zu testen. Überprüfen Sie gegebenenfalls die Einstellungen entsprechend Absatz 6, „Das myAVR Controlcenter“.



8.7 Interrupt-Service-Routine (ISR) im großen Programm

Interrupt-Service-Routinen (im weiteren ISR) sind besondere Formen von Unterprogrammen. Diese werden von einer Interruptquelle des Mikrocontrollers (Timer, ADC, UART, usw.) bei entsprechenden Ereignissen automatisch an beliebiger Stelle im Programmfluss aufgerufen (Unterbrechung, engl. Interrupt). Es ist nötig die Interruptquelle entsprechend zu konfigurieren. Es empfiehlt sich für jedes interruptfähige Gerät eine eigene Unit anzulegen. Diese ist mit der Haupt-Unit zu verbinden.



9 Entwicklung eines Programmablaufplans mit SiSy AVR

9.1 Einleitung

Für die Entwicklung eines Programmablaufplans (PAP) sind konkrete Vorstellungen über die Systemlösung und Kenntnis der Hardware nötig. Ein Programmablaufplan kann aus einer genauen Aufgabenstellung abgeleitet werden.

Beispielaufgabe:
Entwickeln Sie eine Mikrocontrollerlösung, bei der ein Taster eine LED schaltet. Der Controller ist so zu initialisieren, dass an Port B.0 der Taster und an Port B.1 die LED angeschlossen ist. Danach ist fortlaufend der Taster abzufragen. Wenn der Taster gedrückt ist, wird die LED eingeschaltet, sonst bleibt die LED aus.

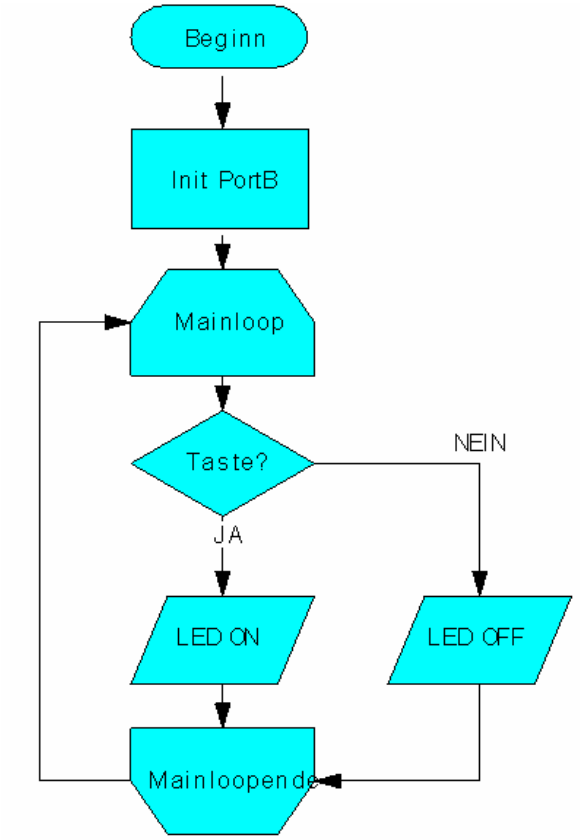
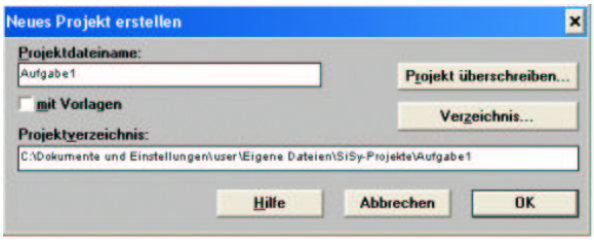
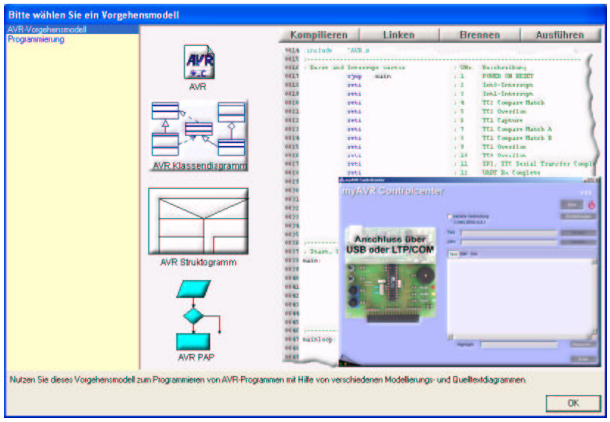


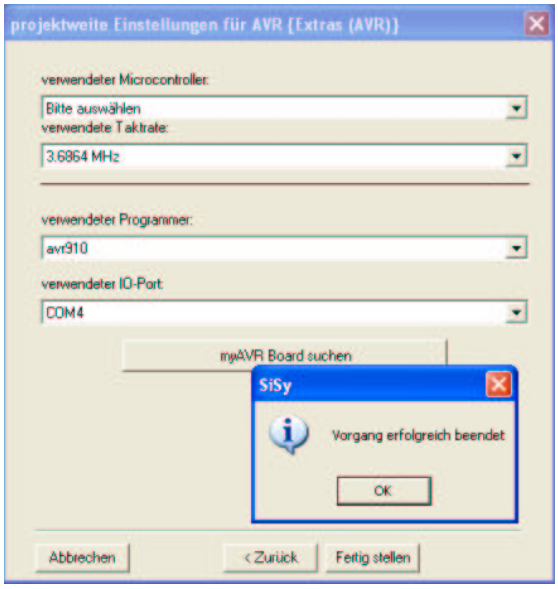
Abbildung 11: PAP zur Beispielaufgabe

9.2 Vorbereitung

Starten Sie SiSy und legen Sie ein neues Projekt an. Wählen Sie das AVR-Vorgehensmodell. Nehmen Sie die Grundeinstellungen für die verwendete AVR Hardware vor oder lassen Sie die myAVR-Hardware automatisch suchen.







Danach öffnet sich die typische Benutzeroberfläche von SiSy mit einem leeren Vorgehensmodell und Sie können mit der Arbeit beginnen. Falls Sie noch die Option „Menü bei Doppelklick“ und „Direkthilfe“ eingeschaltet haben, können Sie diese über den Menüpunkt „Einstellungen“ im Hauptmenü abschalten.

Ziehen Sie als nächstes aus der Objektbibliothek ein Objekt vom Typ „PAP“ in das leere Diagramm. Benennen Sie den PAP mit „Aufgabe 1“. Beachten Sie die Einstellungen zum Controllertyp und Programmieradapter unter „Extras (AVR)“; vgl. Abbildung 12. Den Dialog „Definieren“ erreichen Sie über das Rechte-Maustasten-Menü auf dem Objekt.

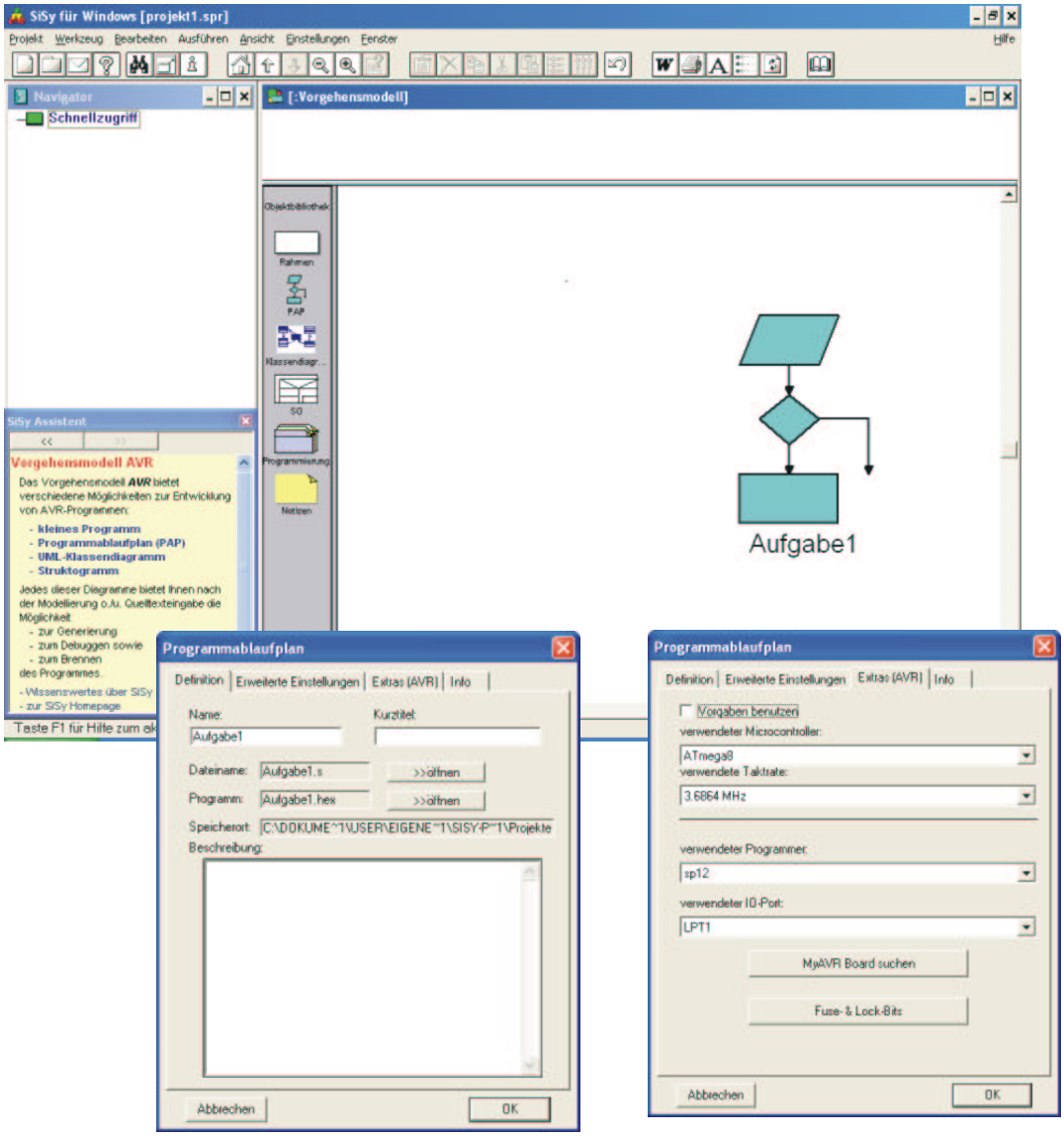
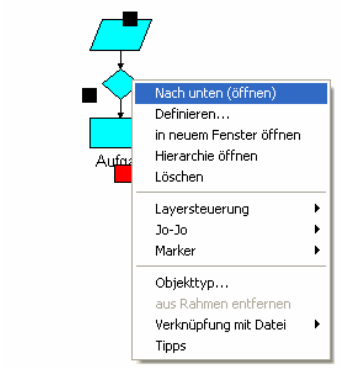


Abbildung 12: Anlegen des Objektes „PAP“ und Einstellungen



Der nächste Schritt ist das Aufstellen des Programmablaufplanes. Dazu muss das Diagramm unter dem Symbol geöffnet werden. Wählen Sie rechte Maustaste „nach unten (öffnen)“, um in dieses Diagramm zu gelangen.

Abbildung 13: PAP öffnen

9.3 Aufgabenstellung

Es ist im ersten Schritt eine einfache Mikrocontroller-Anwendung mit der Technik des Programmablaufplanes zu entwerfen und in der Sprache Assembler zu realisieren.

Aufgabe:
Entwickeln Sie eine Mikrocontrollerlösung, bei der ein Taster eine LED schaltet.

Schaltung:
Port B.0 = Taster 1
Port B.1 = LED

9.4 Grundstruktur laden

Wenn ein Diagramm leer ist, bietet SiSy typische Vorlagen zum Importieren an, siehe Abbildung 15. Diese können dann weiterbearbeitet werden. Wählen Sie die Diagrammvorlage „Mainprogramm-Grundgerüst“.
Abbildung 14 zeigt den PAP zu diesem Grundgerüst.

Abbildung 15: Diagrammvorlagen

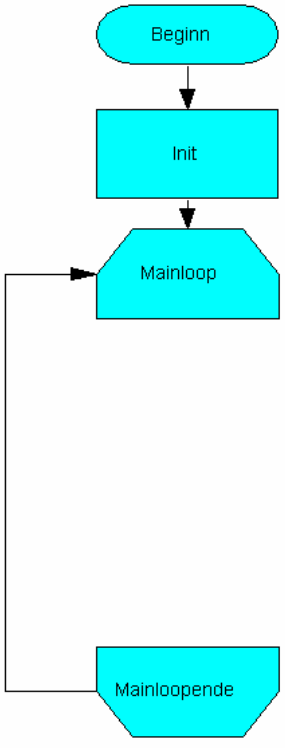


Abbildung 14: Grundgerüst

9.5 Logik entwerfen

Um die Programmlogik im PAP abzubilden, muss die Vorlage um die fehlenden Elemente ergänzt werden. Des Weiteren sind die Elemente durch gerichtete Verbindungen (Kanten) in der Reihenfolge ihrer Abarbeitung zu verbinden.
Um ein Objekt im Diagramm zu ergänzen, wird der entsprechende Objekttyp in der Objektbibliothek mit der Maus ausgewählt und per Drag & Drop an die entsprechende Position im Diagramm gezogen. Objekte können mit dem Mauscursor per Click selektiert und mittels Drag & Drop auch verschoben werden. Selektierte Objekte lassen sich mit der Taste „Entf“ löschen. Verbindungen zwischen den Objekten können über den rot markierten „Verteiler“ von selektierten Objekten hergestellt werden. Dazu ist das Ausgangsobjekt zu selektieren mit dem Mauscursor auf den roten Verteiler zu klicken und bei gedrückter linker Maustaste eine Verbindung zum Zielobjekt zu ziehen. Um Objekte zu benennen, können Sie einen Doppelklick auf dem betreffenden Objekt durchführen oder über rechte Maustaste in den Dialog *Definieren* gelangen.

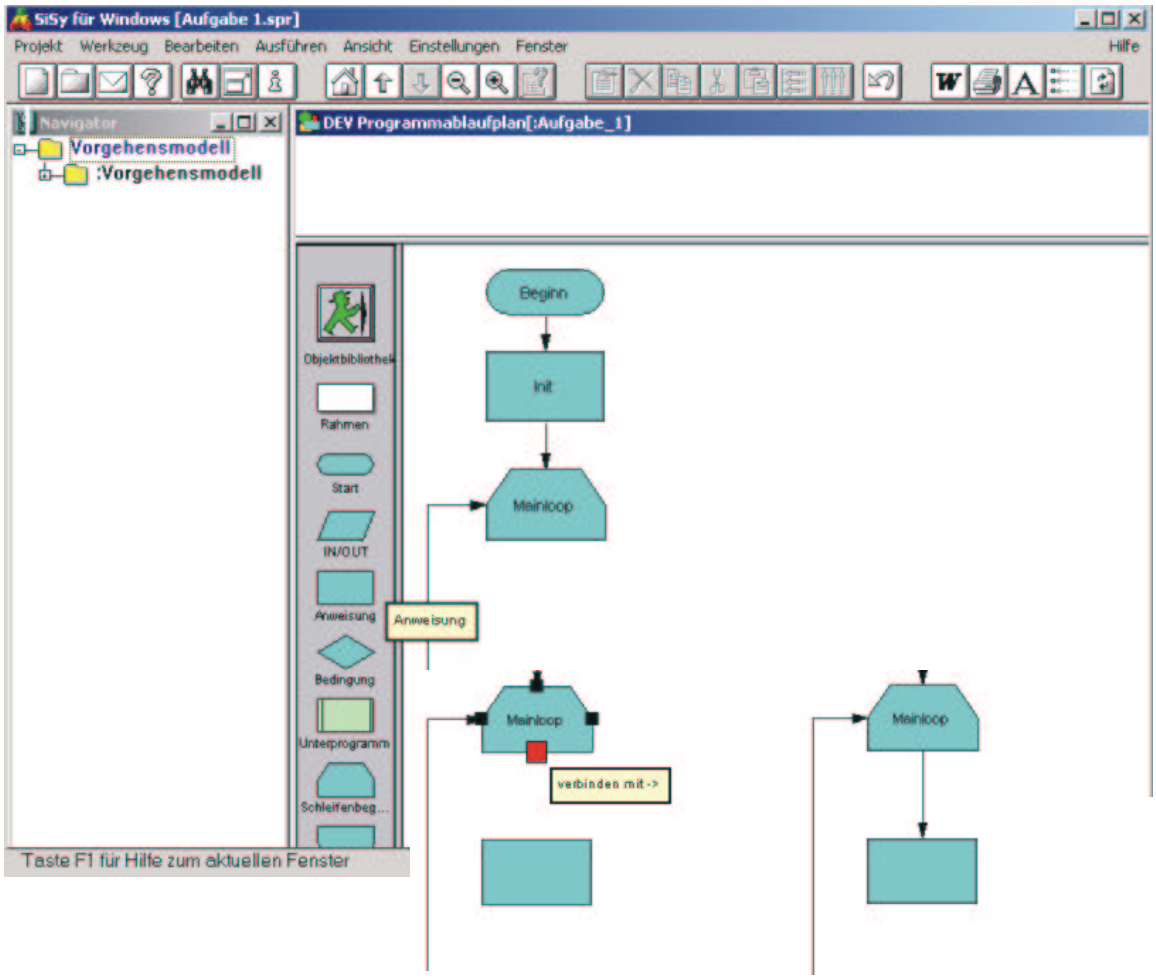


Abbildung 16: Objekte in das Diagramm einfügen und verbinden

Zeichnen Sie den folgenden Programmablaufplan (vgl. Abbildung 17):

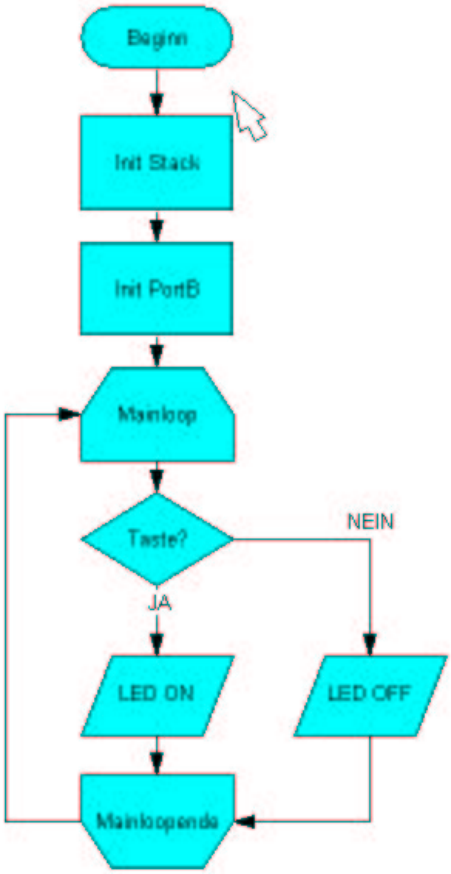


Abbildung 17: Logikentwurf im PAP

9.6 Befehle eingeben

Nachfolgend soll aus dem Programmablaufplan Assemblerquellcode generiert werden. Dazu ist es nötig, die einzelnen Elemente des PAP mit den entsprechenden Assembleranweisungen zu versehen. Dafür gibt es mehrere Möglichkeiten.

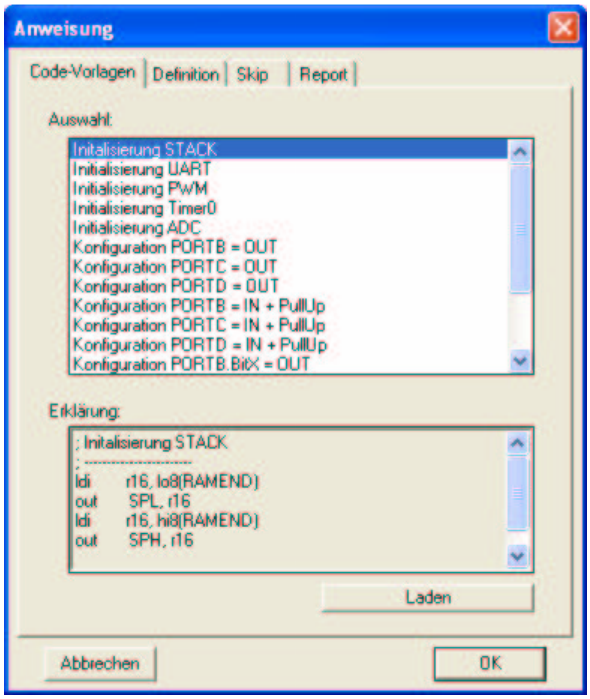


Abbildung 18: Codevorlage PAP

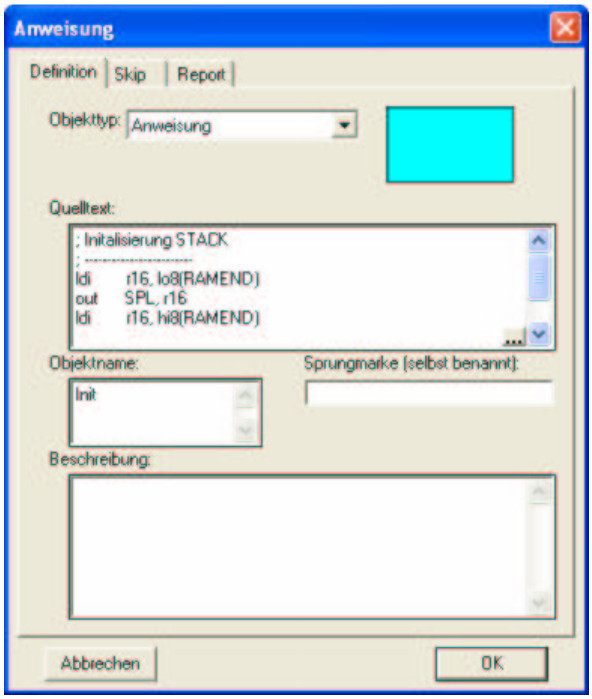


Abbildung 19: Definition PAP-Element

Zum einen bietet SiSy beim ersten Öffnen eines jeden Elementes typische Code-Vorlagen an, die über die Schaltfläche „Laden“ dem Element zugewiesen werden können. Wird der Definieren-Dialog mit der Schaltfläche „OK“ beendet, so wird die Auswahl im Objekt gespeichert und beim nächsten Aufruf des Dialoges „Definieren“ erscheinen die Code-Vorlagen nicht mehr, sondern das Element kann ganz normal bearbeitet werden. In Abbildung 18 und Abbildung 19 sind beide Varianten des Dialoges „Definieren“ zu sehen“.

Die zweite Möglichkeit besteht beim Selektieren von Elementen über den Quellcodeeditor oberhalb des Diagrammfensters, vgl. Abbildung 21. und Abbildung 20

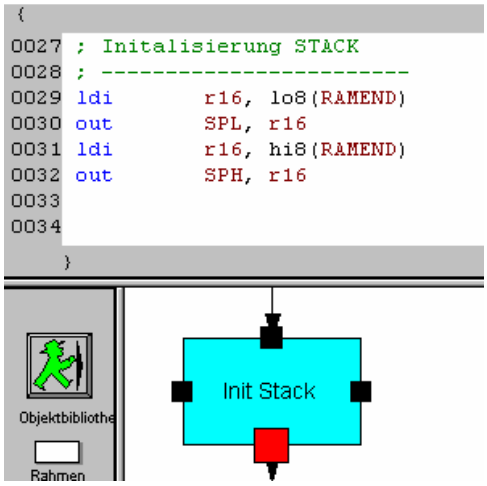


Abbildung 21: Quellcodefenster im PAP

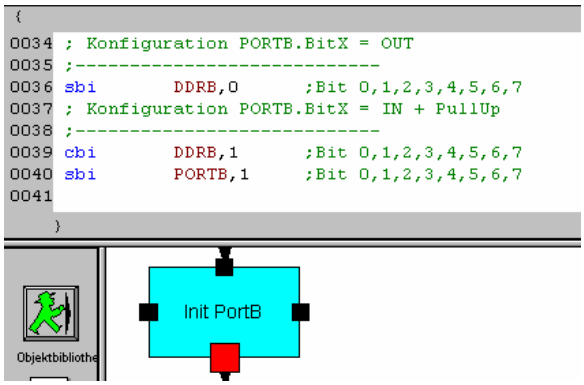


Abbildung 20: Quellcodeeingabe PAP

Geben Sie die gezeigten Quellcodes in die Objekte ein!

Bedingungen haben spezielle Vorlagen, die eine Codegenerierung bei übersichtlichem Programmablaufplan vereinfachen.
Jede Bedingungsvorlage ist so konstruiert, dass eine JA/NEIN Entscheidung erzeugt werden kann. Findet der Codegenerator das Schlüsselwort JA an einer der folgenden Verbindungen, setzt er diese in eine Sprunganweisung „breq“ um. Das Schlüsselwort NEIN wird in „brne“ umgewandelt. Alternativ können statt dieser Schlüsselworte auch der Sprungbefehl selber an eine der Kanten geschrieben werden (breq, brne, brge, brlo, usw.)

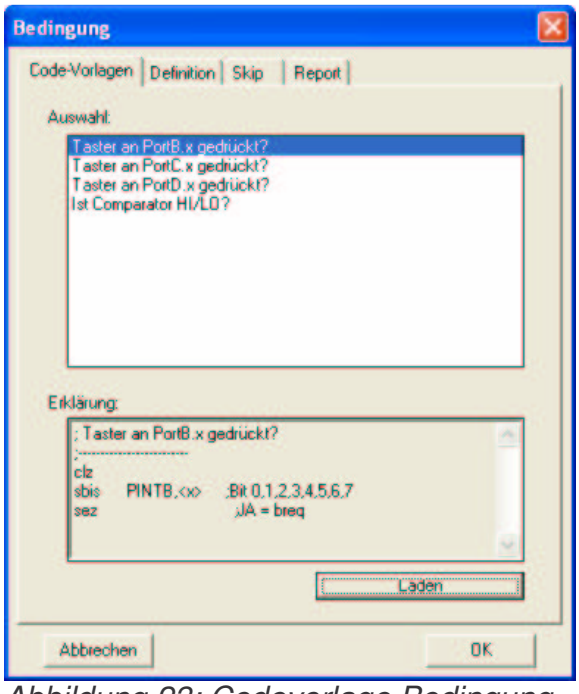


Abbildung 23: Codevorlage Bedingung

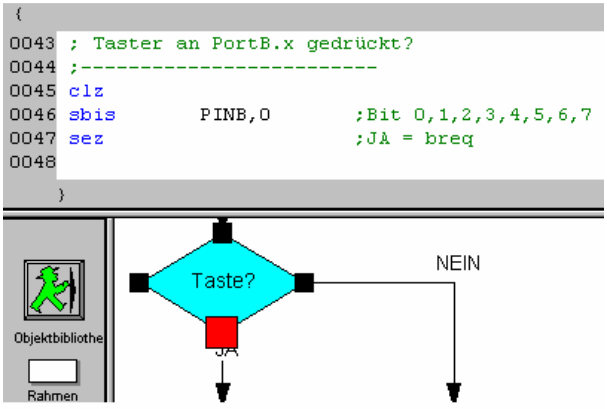


Abbildung 22: Quellcode einer Bedingung

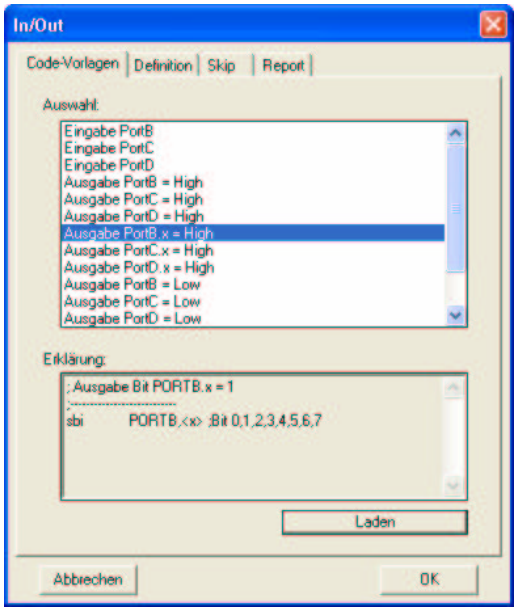
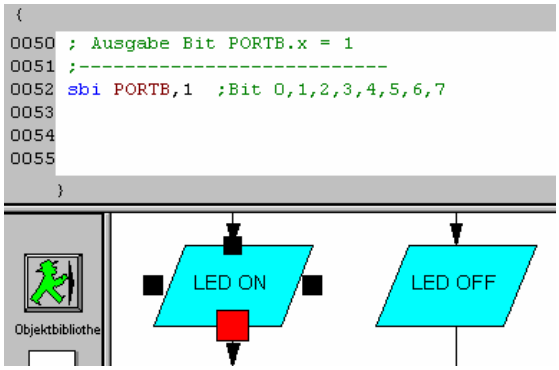
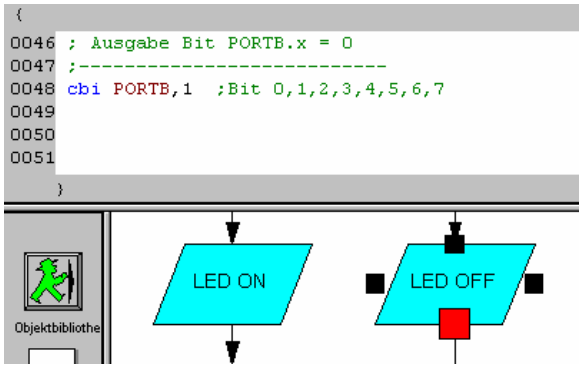


Abbildung 24:Vorlagen für IN/OUT

Auch das Element „IN/OUT“ verfügt über spezifische Vorlagen. Diese sind gegebenenfalls mit zu ergänzen. Dazu sind spitze Klammern als Platzhalter in den Vorlagen eingefügt.

Ergänzen Sie den Quellcode der gezeigten Elemente!





9.7 Übersetzen, Brennen und Test

Sind alle betreffenden Elemente mit Quellcode hinterlegt, kann aus dem Programmablaufplan der komplette Quellcode generiert, kompiliert, gelinkt und auf den Mikrocontroller übertragen werden. Die gewünschte Funktion kann aus dem Aktionsmenü ausgewählt werden (Abbildung 25).

Hinweis:

Beachten Sie, dass für das Brennen des Controllers das Programmierkabel angeschlossen sein muss und bei Bedarf eine geeignete Spannungsquelle anzuschließen ist.

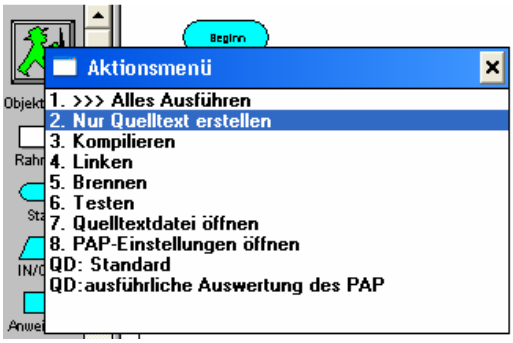


Abbildung 25: Auswahl aus Aktionsmenü

Funktionalitäten des Aktionsmenüs	
Benennung	**Funktion**
Alles Ausführen	Quellcode generieren, kompilieren, linken, brennen
Nur Quelltext erstellen	Quellcode generieren mit allen Marken
Kompilieren	Quellcode zu Objektdatei übersetzen
Linken	Objektdatei zu HEX-Datei binden
Brennen	HEX-Datei an den Controller übertragen
Testen	myAVR Controlcenter öffnen
Quelltextdatei öffnen	Quellcodedatei öffnen
Quelltext bereinigen	Quellcode von überflüssigen Marken bereinigen

Im Ausgabefenster (siehe Abbildung 26) werden die jeweiligen Aktionen angezeigt.

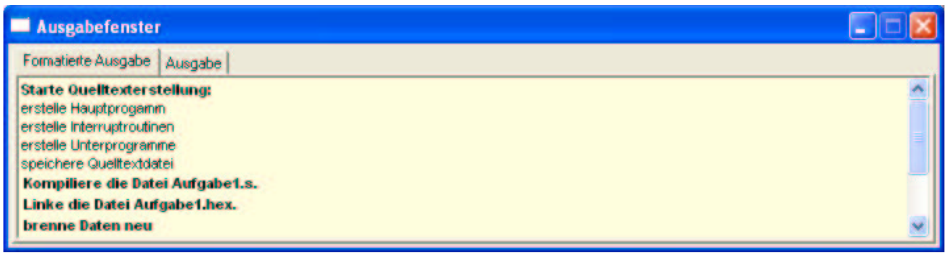


Abbildung 26: Ausgabefenster

Bei Compilerfehlern werden diese ebenfalls im Ausgabefenster mit der entsprechenden Zeilennummer angezeigt. Um zu dem Fehler zu gelangen, genügt meist ein Klick auf die Fehlermeldung. Das betreffende Objekt wird selektiert und die Zeile hervorgehoben.

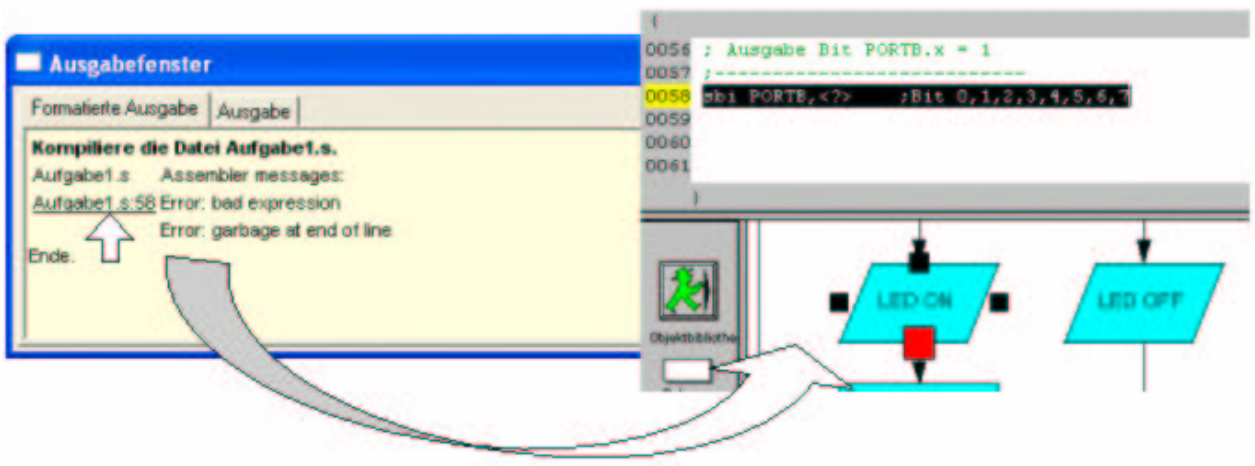


Abbildung 27: Fehlerbehandlung

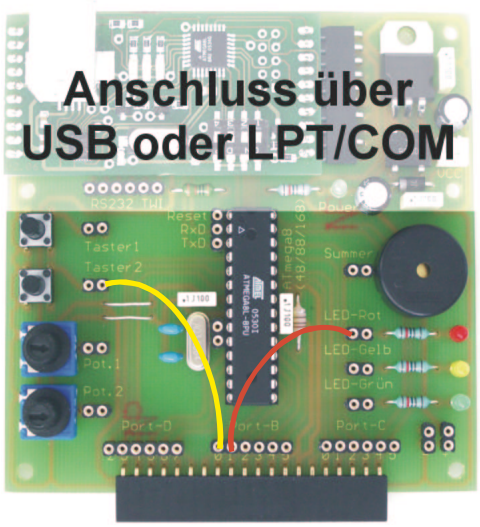


Abbildung 28

Nachdem das Programm erfolgreich übersetzt und auf den Controller übertragen wurde, kann die Anwendung getestet werden. Dazu ist das myAVR-Controlcenter zu öffnen und die vorgegebenen Verbindungen auf dem Board zu stecken. Über die Schaltfläche „Start/Stopp“ kann das myAVR Board ein und ausgeschaltet werden.

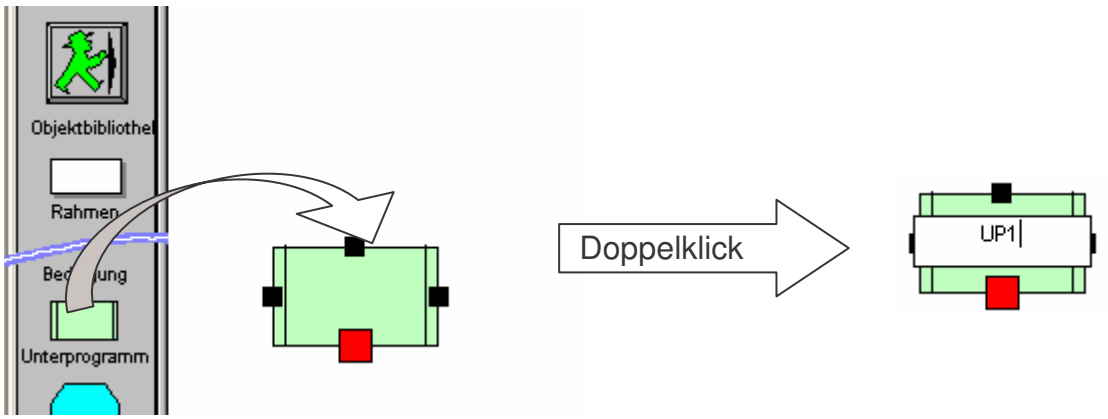
Abbildung 29

9.8 Unterprogrammtechnik im PAP

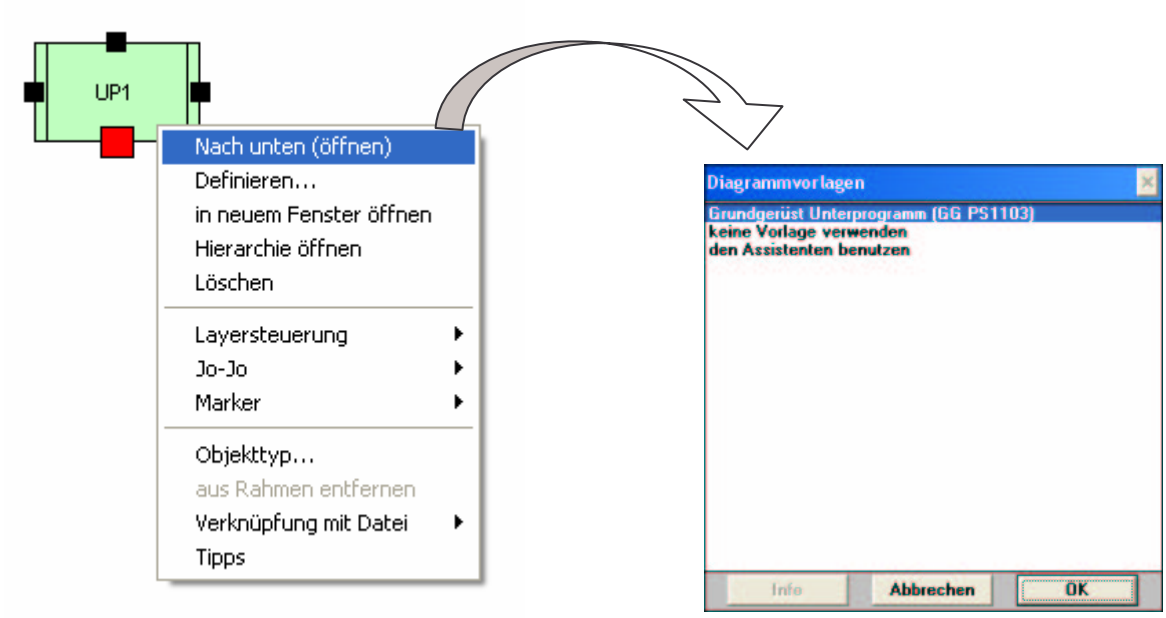
Unterprogramme sind ein wichtiges Gestaltungsmittel für übersichtliche Mikrocontrollerprogramme. Sie werden für in sich abgeschlossene Aufgaben (Verarbeitungsschritte) benutzt, die auch mehrfach im Gesamtprogramm genutzt werden können.

9.8.1 Anlegen eines Unterprogramms

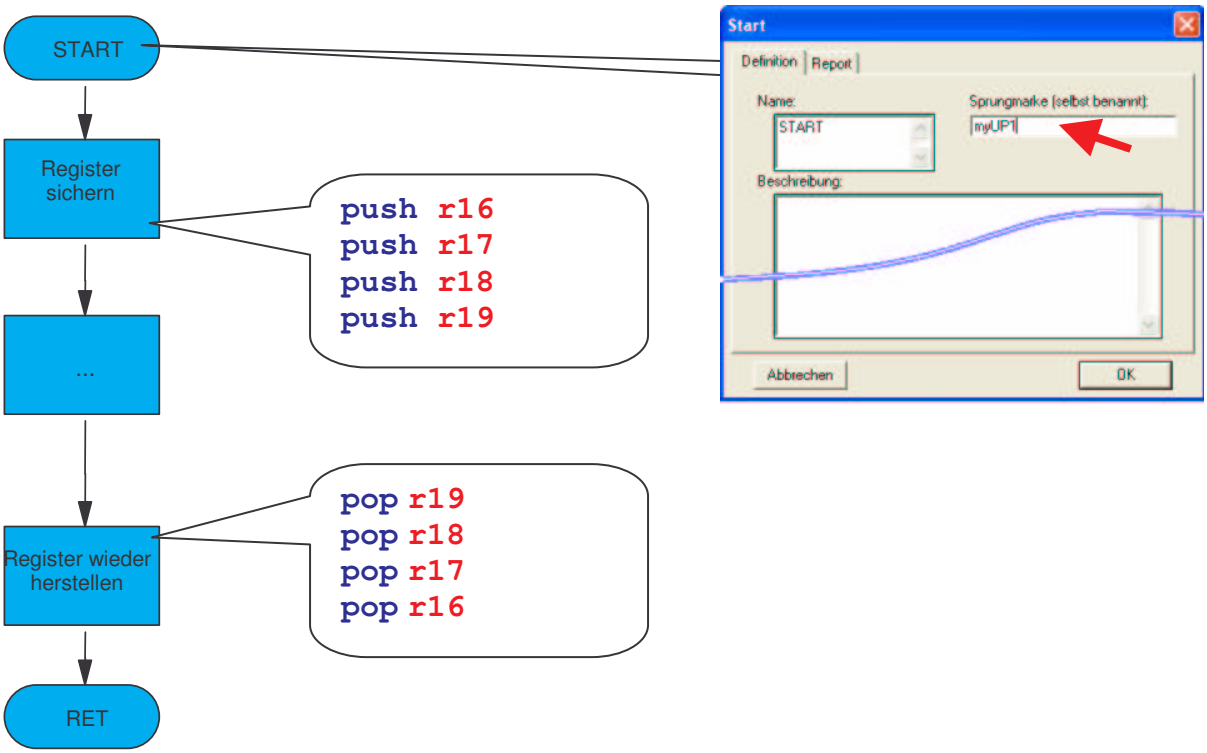
Ziehen Sie den Objekttyp „Unterprogramm“ aus der Objektbibliothek in das gewünschte Diagramm. Mit Doppelklick oder über rechte Maustaste -> Kontextmenü -> Definieren auf dem Element können Sie dem Unterprogramm einen Namen geben (Beachte: Menüfolge Einstellungen/Menü bei Doppelklick).



Damit ist ein Objekt angelegt, welches im aktuellen Diagramm als Aufruf (call) des Unterprogramms zu verstehen ist. Die Funktionalität des Unterprogramms wird in einem gesonderten Programmablaufplan für das Unterprogramm entworfen. Dazu ist das Diagramm „unter“ bzw. „hinter“ Unterprogramm zu öffnen. Um das zum Unterprogramm zugehörige Diagramm zu öffnen, wählen Sie auf dem Objekt rechte Maustaste -> Kontextmenü -> Nach unten (öffnen).

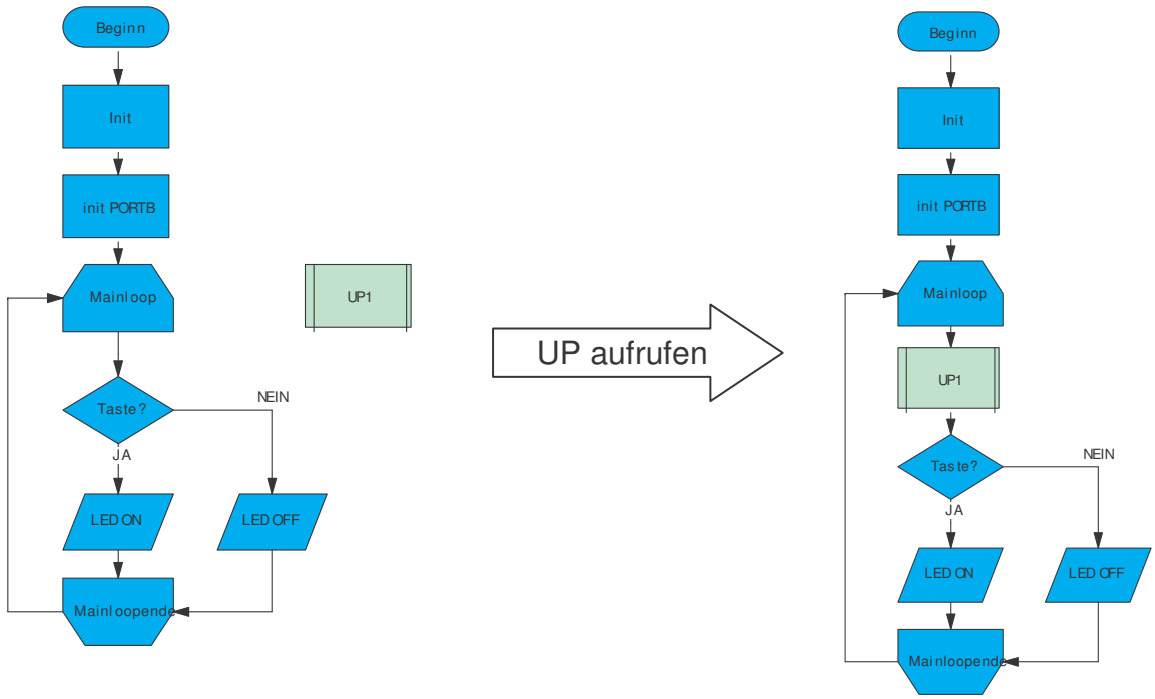


Sie erhalten eine Vorlagenliste für die Grundstruktur von Unterprogrammen. Bitte laden Sie die Vorlage „Grundgerüst Unterprogramm“. Auf dem Objekt „START“ können Sie eine benutzerdefinierte Sprungmarke festlegen, die durch den Codegenerator erstellt und verwendet werden soll. Die Vorlage muss entsprechend der vorgesehen Logik abgeändert werden.



9.8.2 Ein Unterprogramm aufrufen

Das Unterprogrammsymbol muss zum Aufruf an der entsprechenden Position im Programmablaufplan eingefügt werden. Der Codegenerator erzeugt dann entsprechend einen Unterprogrammaufruf und den Code für das Unterprogramm selbst. Dazu ist in das Diagramm zurückzukehren, in dem das Symbol „Unterprogramm“ angelegt wurde (zum Beispiel rechte Maustaste -> Kontextmenü-> nach oben…). Das Unterprogramm ist korrekt eingebunden, wenn es vollständig und eindeutig im Programmfluss integriert ist (mindestens ein eingehender Pfeil und genau ein ausgehender Pfeil).



9.8.3 Unterprogramme mehrmals benutzen

Ein wesentliches Merkmal von Unterprogrammen ist, dass diese von verschiedenen Stellen im Programm aufgerufen (call) werden können und auch dorthin zurückkehren (return). Um diese Möglichkeit zu nutzen, bietet SiSy das Anlegen von Referenzen an. Vergleichen Sie dazu Absatz 3.2 „Die Modellierungselemente von SiSy“. Um ein Unterprogramm zu referenzieren (wiederholend zeigen und einbinden) gehen Sie wie folgt vor:

1. zeigen Sie im Navigator das gewünschte Unterprogramm an
   a. über den Schnellzugriff,
      dort lässt sich das Original per Drag & Drop ablegen, oder
   b. über die Navigatorsortierung „Unterprogramme“
      Navigator -> rechte Maustaste -> Kontextmenü -> Unterprogramme
2. das gewünschte Zieldiagramm, in dem das Unterprogramm verwendet werden soll, öffnen
3. per Drag und Drop das Unterprogramm in das Zieldiagramm ziehen, dabei wird nur eine Referenz (Link) auf das Original angelegt.
4. Referenz wie oben beschrieben in den Programmfluss integrieren

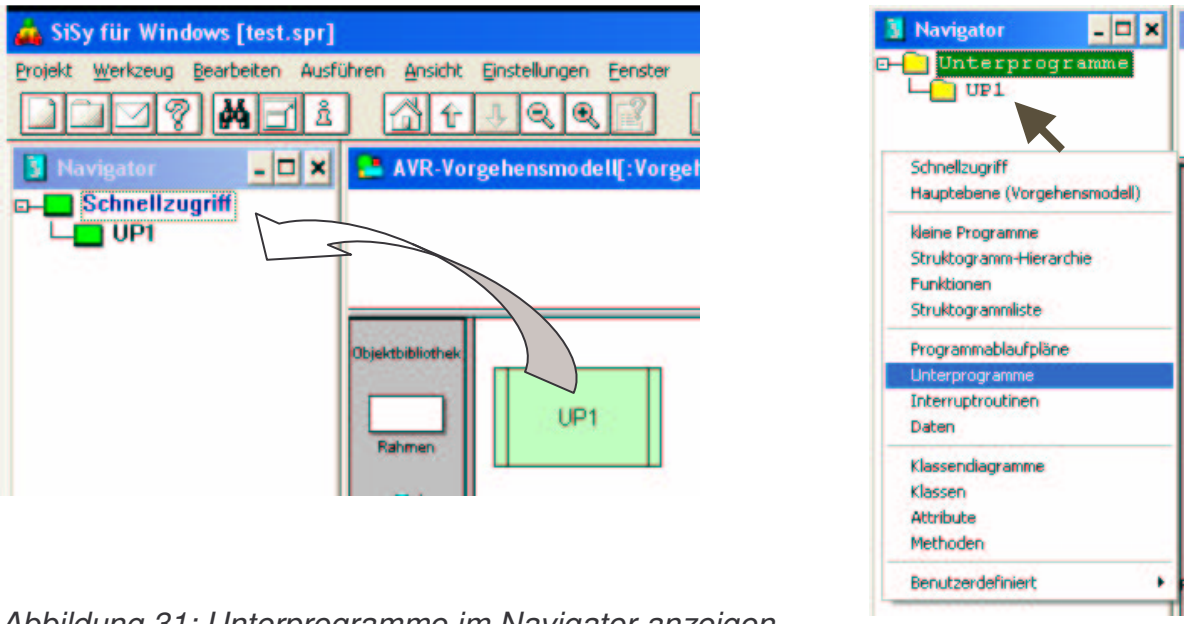


Abbildung 31: Unterprogramme im Navigator anzeigen

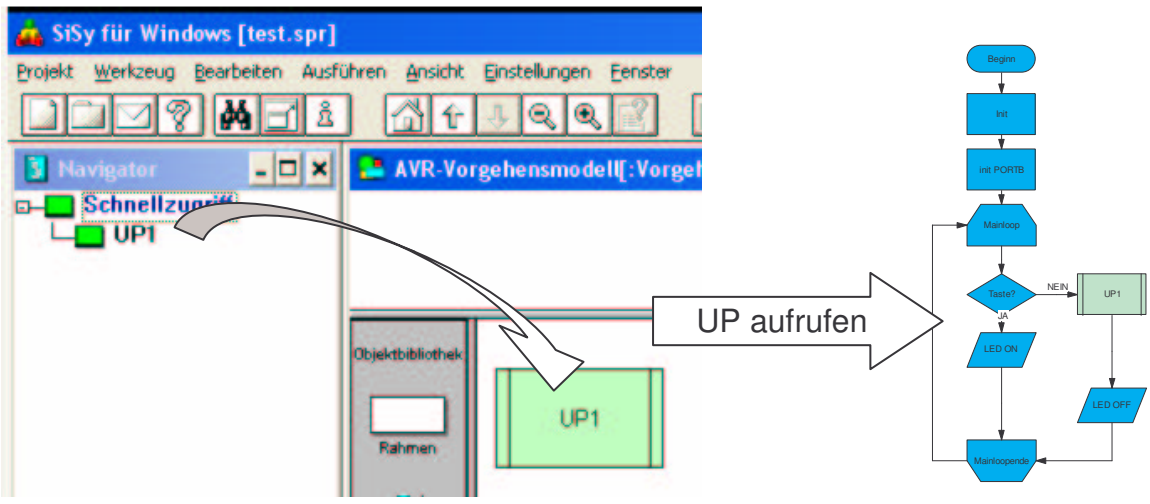
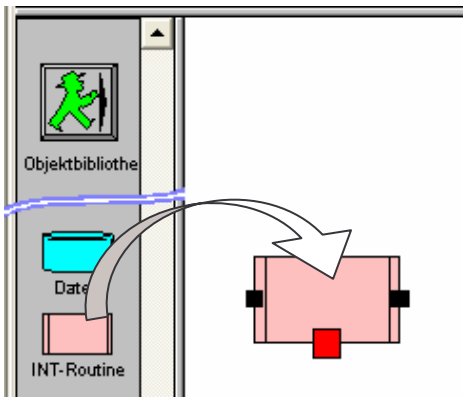
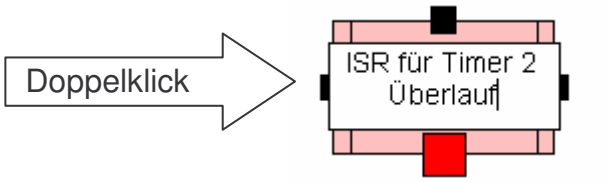


Abbildung 30: Unterprogramm referenzieren

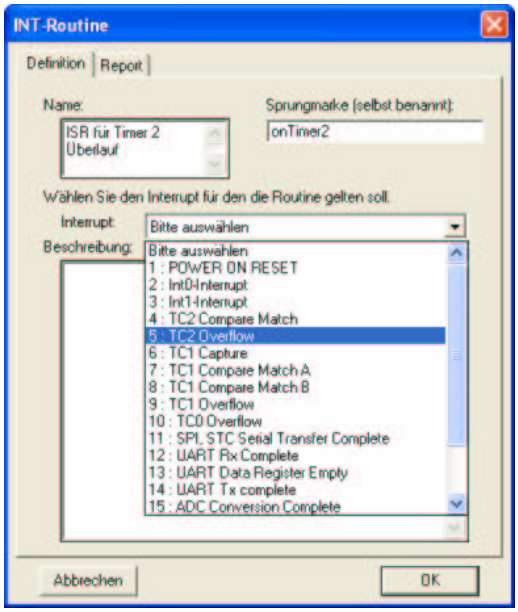
9.9 Interrupt-Service-Routinen (ISR) im PAP

Interrupt-Service-Routinen (im weiteren ISR) sind besondere Formen von Unterprogrammen. Diese werden von einer Interruptquelle des Mikrocontrollers (Timer, ADC, UART, usw.) bei entsprechenden Ereignissen automatisch an beliebiger Stelle im Programmfluss aufgerufen (Unterbrechung, engl. Interrupt). Es ist demzufolge nicht vorgesehen, eine ISR in den Programmfluss zu integrieren.
Um eine ISR zu erzeugen, ziehen Sie ein Objekt vom Typ INT-Routine aus der Objektbibliothek in das Diagramm des Hauptprogramms und definieren einen Namen.

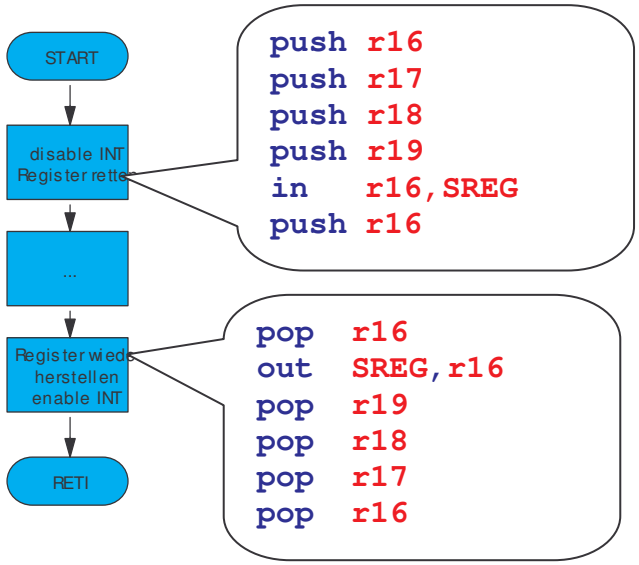


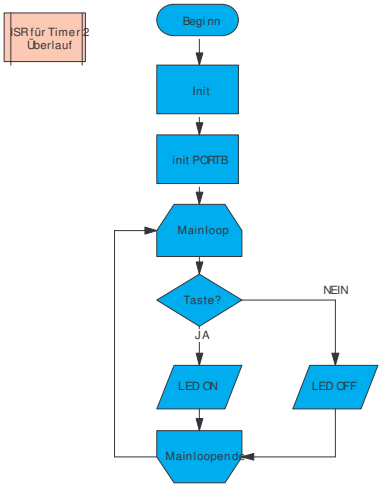


Über rechte Maustaste -> Kontextmenü -> Definieren … legen Sie die Sprungmarke und den Typ des Interrupt fest. Damit erfolgt durch den Codegenerator die Zuordnung der ISR zum entsprechenden Interruptvektor. Beachten Sie, dass die Liste der Interrupts abhängig ist vom gewählten Controllertyp.



Zum Entwerfen der ISR-Logik wählen Sie auf dem Objekt rechte Maustaste -> Kontextmenü -> nach unten … Ihnen wird das Grundgerüst einer ISR als Diagrammvorlage angeboten. Laden Sie die Diagrammvorlage.
Vervollständigen Sie danach die ISR-Logik. Die ISR wird nicht in den Programmfluss integriert.



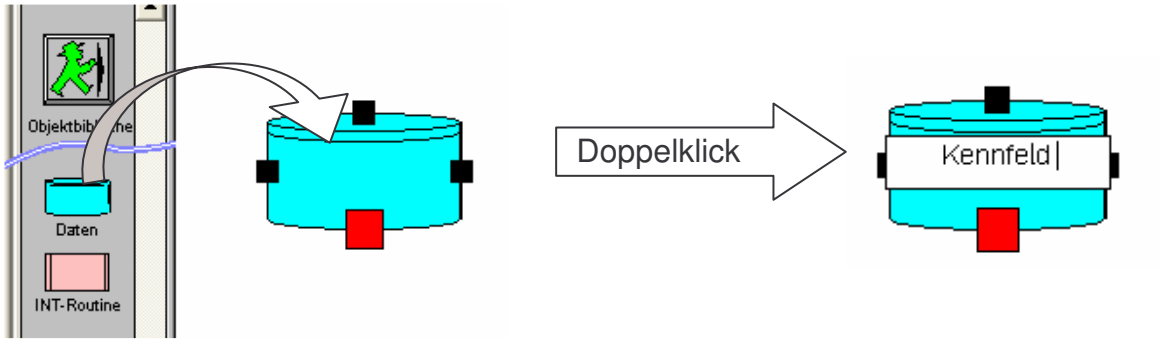


9.10 Daten im PAP

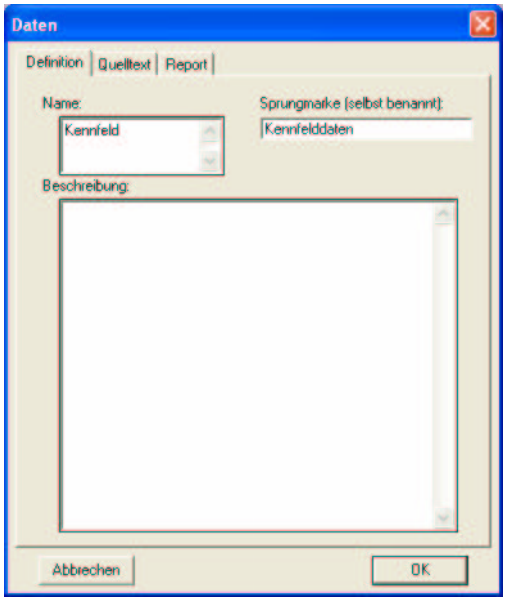
Konstante Daten, die im Programmspeicher (FLASH) des Mikrocontrollers abgelegt werden, können als Datenobjekt deklariert werden. Die im Datenobjekt deklarierten Daten werden durch den Codegenerator immer ans Ende des Quellcodes gestellt. Die zu generierende Marke/Marken für die Datenelemente können vom Entwickler frei gewählt werden.

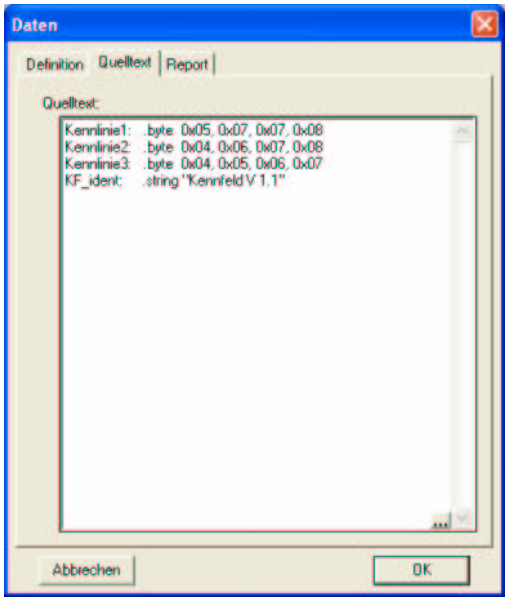
9.10.1 Anlegen eines Datenobjektes

Zum Anlegen eines Datenobjektes ziehen Sie das betreffende Objekt per Drag & Drop aus der Objektbibliothek in das gewünschte Diagramm. Per Doppelklick können Sie einen Namen vergeben (Beachte: Menüfolge *Einstellungen/Menü bei Doppelklick*).



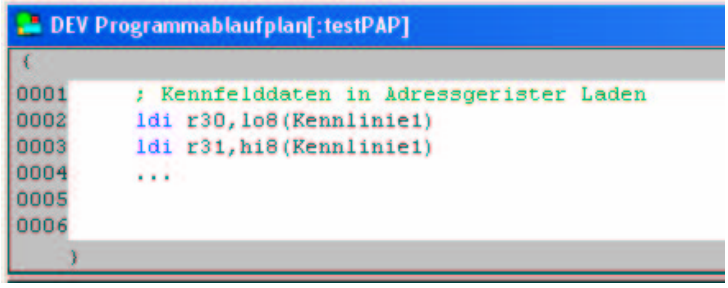
Die Marke und die Daten selbst können über rechte Maustaste -> Kontextmenü -> Definieren… festgelegt werden.





9.10.2 Datenobjekt benutzen

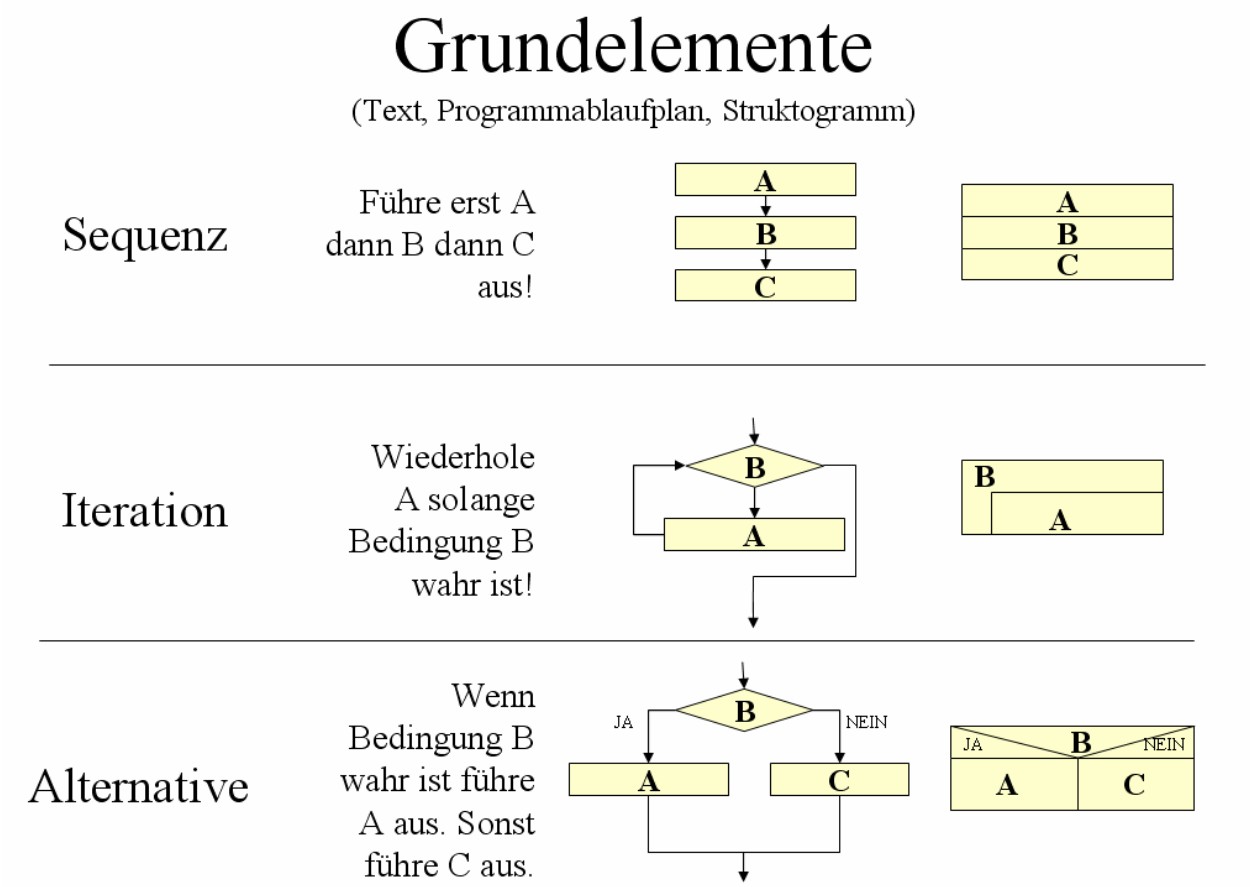
Im Quellcode werden die Daten über die vergebenen Markennamen angesprochen.



10 Entwicklung eines Struktogramms mit SiSy AVR

10.1 Einleitung

Struktogramme (SG) oder auch Nassi-Shneiderman-Diagramme sind ein Entwurfsmittel der strukturierten Programmierung. Strukturiert meint in dem Fall, dass zur Realisierung eines Algorithmus auf das Verwenden von Sprunganweisungen (Goto, Jump) verzichtet wird. Für das Formulieren eines Algorithmus stehen dem Entwickler drei normierte Grundstrukturen zur Verfügung: Sequenz (Folge von Anweisungen), Alternative (Auswahl bzw. bedingte Anweisung), Iteration (Schleife, wiederholte Anweisung).



Struktogramme werden als Darstellungsmittel für strukturierte Sprachen wie C oder PASCAL verwendet, da hier im Gegensatz zu Assembler in der Regel auf Sprunganweisungen verzichtet wird.

10.2 Aufgabenstellung

Um die Arbeitsweise und den Umgang mit dem Struktogrammeditor in SiSy zu erlernen, realisieren Sie bitte das folgende Beispiel.

Es soll eine Mikrocontrollerlösung entwickelt werden, bei der auf Tastendruck eine LED eingeschaltet wird. Dabei soll der Taster an Port B 0 und die LED an Port B 1 angeschlossen werden

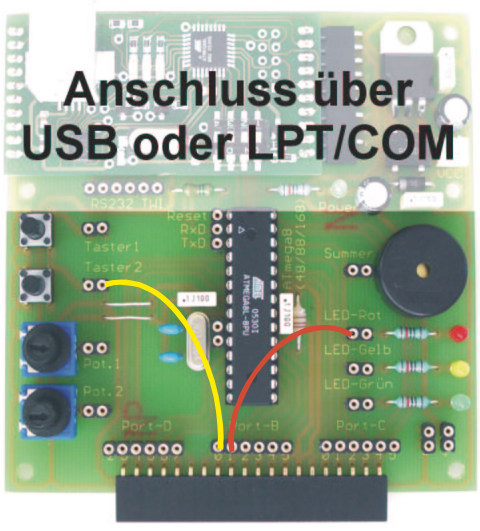
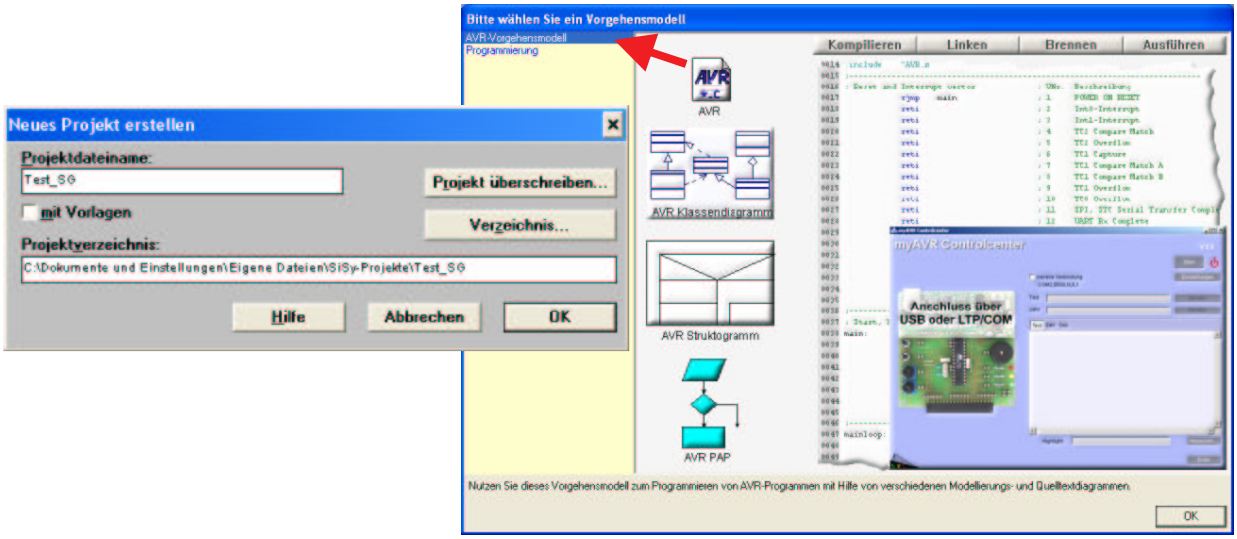


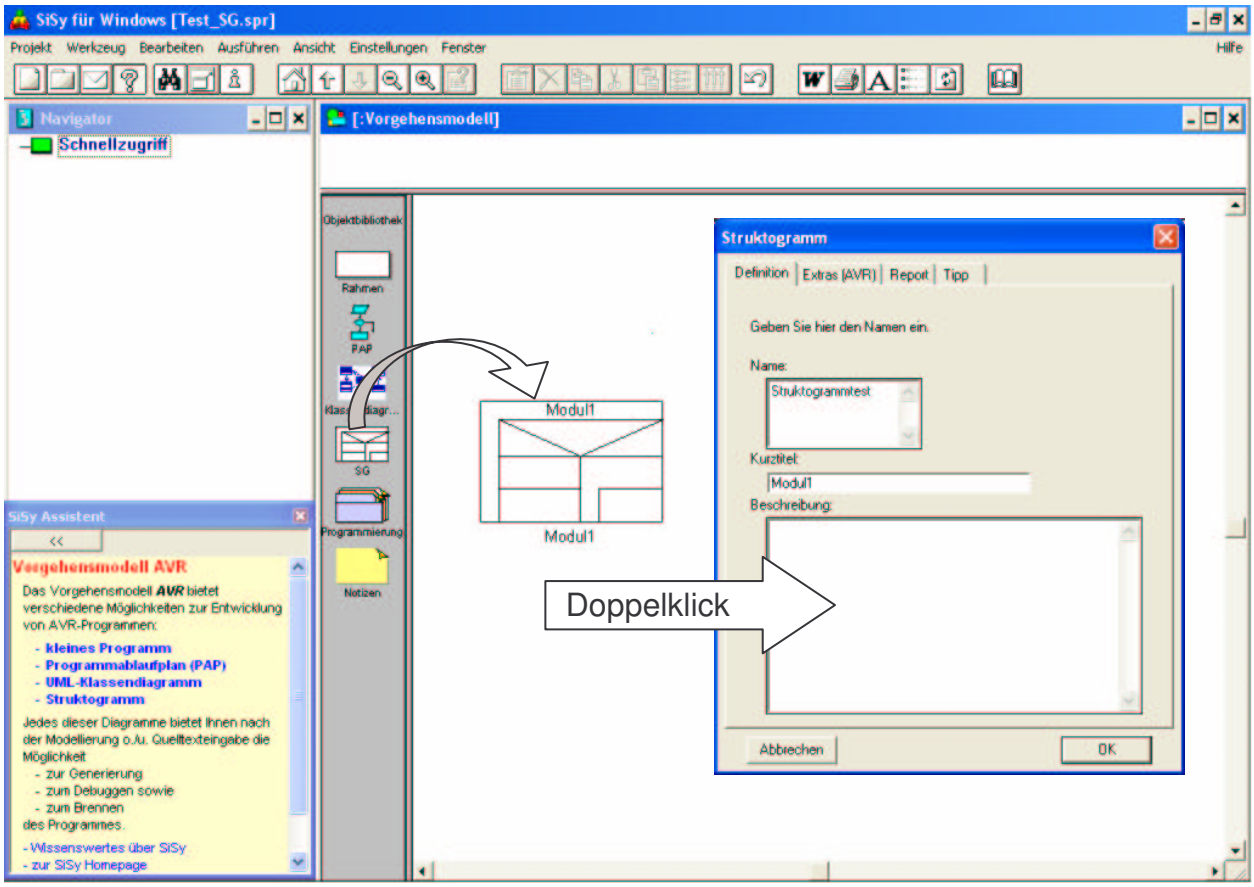
Abbildung 32: Beispiel für SG

10.3 Vorbereitung

Das Struktogramm kann je SiSy Projekt ein Hauptprogramm verwalten. Es ist also nötig, für jedes neue Struktogramm-Projekt auch ein neues SiSy-Projekt zu erzeugen. Starten Sie ggf. SiSy.
Erstellen Sie ein neues SiSy-Projekt (Hauptmenu -> Projekt -> Neu) mit dem Namen „Test_SG“. Wählen Sie das Vorgehensmodell „AVR-Vorgehensmodell“.

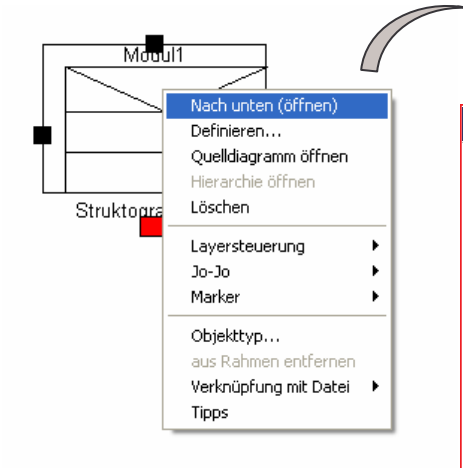


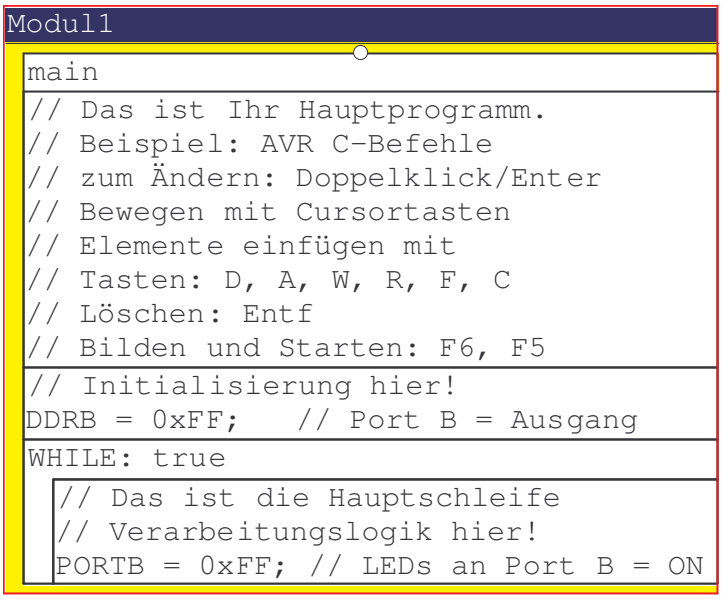
Legen Sie im weiteren Verlauf die AVR - Grundeinstellungen für das Projekt fest. Ziehen Sie ein Objekt vom Typ SG (Struktogramm) per Drag & Drop in das Diagrammfenster. Legen Sie den Namen für das Struktogramm fest.



10.4 Struktogramm entwickeln

Um das Struktogramm zu entwickeln, muss das Struktogrammfenster geöffnet werden. Wählen Sie dazu auf dem Objekt rechte Maustaste -> Kontextmenü -> nach unten (öffnen).




Modul1
main
// Das ist Ihr Hauptprogramm.
// Beispiel: AVR C-Befehle
// zum Ändern: Doppelklick/Enter
// Bewegen mit Cursortasten
// Elemente einfügen mit
// Tasten: D, A, W, R, F, C
// Löschen: Entf
// Bilden und Starten: F6, F5
// Initialisierung hier!
DDRB = 0xFF; // Port B = Ausgang
WHILE: true
// Das ist die Hauptschleife
// Verarbeitungslogik hier!
PORTB = 0xFF; // LEDs an Port B = ON

Es wird beim ersten Struktogramm im Projekt automatisch die Grundstruktur eines AVR C Programms erzeugt diese kann bereits gebildet, gebrannt und gestartet werden.

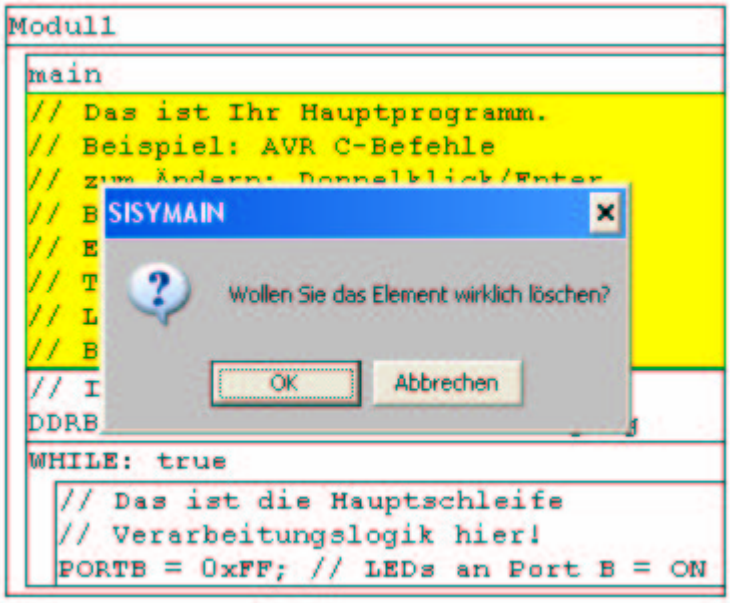
Es empfiehlt sich im Struktogramm mit der Tastatur zu arbeiten.

Tastenbelegung (Auszug) im Struktogramm:

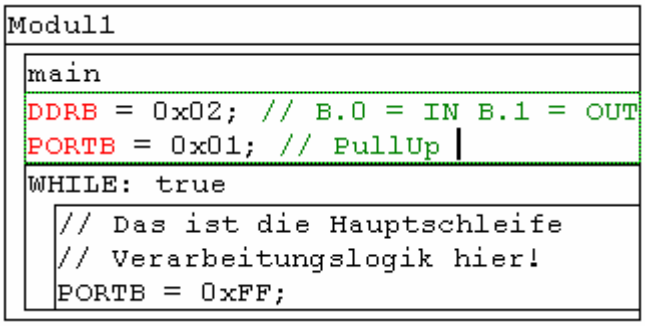
F3	öffnet den Quelltext
F5	startet das Programm (bei SiSy AVR das myAVR Controlcenter)
F8	Codegenerierung des aktuellen Struktogramms
F9	übersetzt das Struktogramm
ESC	Ebene nach oben
Num `+`	selektiertes Objekt wird vergrößert
Num `-`	selektiertes Objekt wird verkleinert
Enter	ermöglicht die Bearbeitung des Quelltextes im Struktogramm-Editor
Entf	löschen
T	ermöglicht die Bearbeitung des Titels
K	ermöglicht die Bearbeitung des Kommentars
B	Begin neue Funktion einfügen
S	Sequenz einfügen (Folge/Block)
D	Do einfügen (elementare Aktion)
W	While-Schleife einfügen (kopfgesteuerte Schleife)
R	Repeat-Schleife einfügen (fußgesteuerte Schleife)
F	For-Schleife einfügen (Zählschleife)
A	Alternative einfügen
C	Case einfügen (Fallunterscheidung)
I	If-Zweig einfügen
E	Exit einfügen
L	Loop einfügen
Cursortasten	Bewegung im Struktogramm

Führen Sie folgende Arbeitsschritte aus, um den Algorithmus für die oben genannte Aufgabe zu entwerfen:

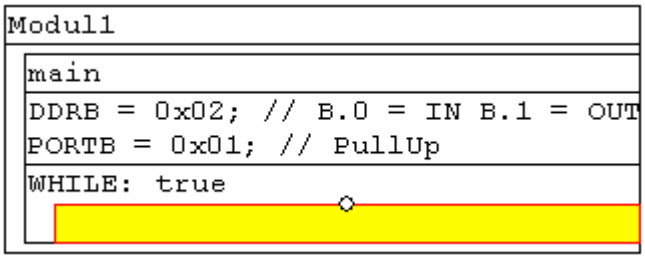
1. Löschen Sie den Kommentar, selektieren Sie dazu das Element zum Beispiel mit der Maus oder den Cursortasten. Betätigen Sie dann die Taste „Entf“ und bestätigen Sie die Sicherheitsabfrage mit „OK“.



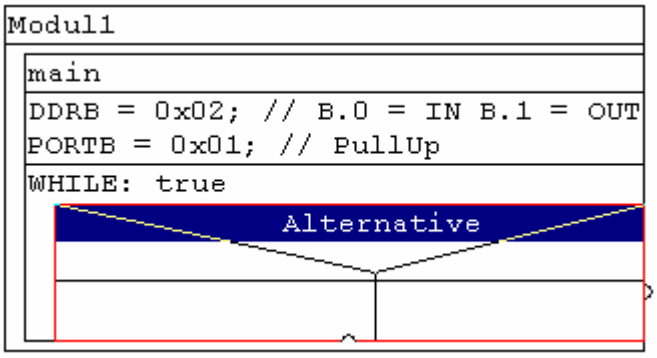
2. Konfigurieren Sie Port B 0 als Eingang und Port B 1 als Ausgang. Selektieren Sie dazu die Initialisierungssequenz. Mit ENTER kommen Sie in den Editiermodus. Mit ESC verlassen Sie den Editiermodus.



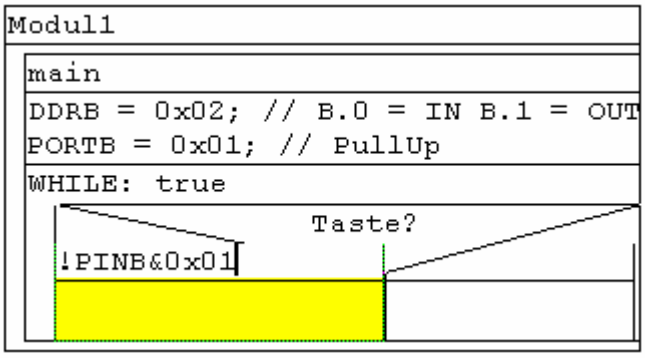
3. Löschen Sie das DO-Element in der Mainloop (WHILE-Schleife). Dazu selektieren Sie mit der Maus oder den Cursortasten das DO-Element und betätigen die Taste „Entf“.



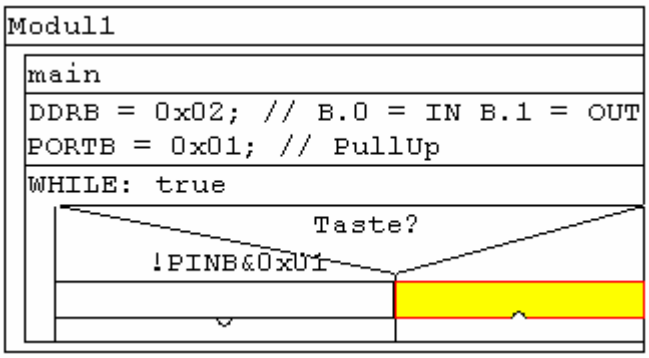
4. Ergänzen Sie die Mainloop durch eine Alternative. Dazu betätigen Sie die Taste „A“. Achten Sie darauf, dass die Alternative innerhalb der Mainloop liegt.



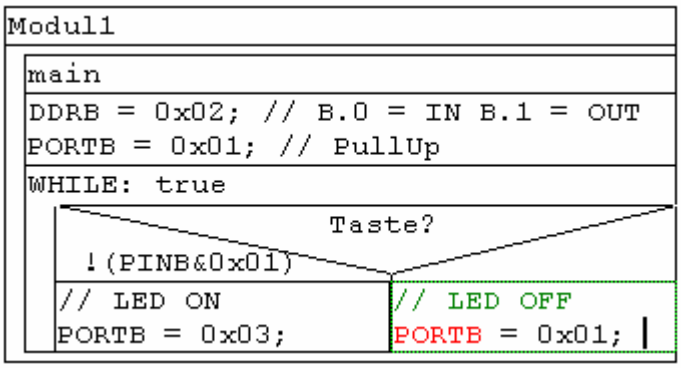
5. Geben Sie den Titel und die Bedingung für die Alternative ein. Dazu selektieren Sie das gewünschte Element und betätigen die Taste ENTER oder Doppelklick mit der Maus.

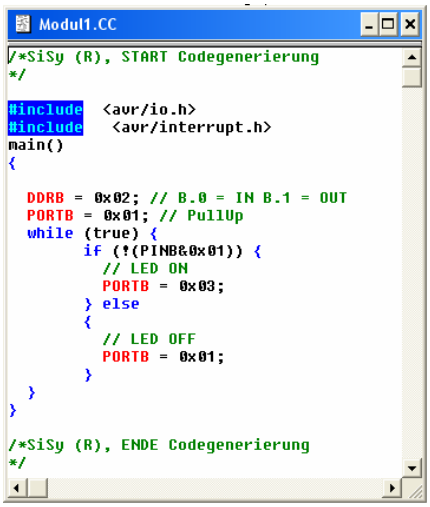


6. als nächstes sind die Aktionen (DO-Elemente) in der Alternative zu ergänzen. Selektieren Sie dazu den Alternativzweig mit der Maus oder den Cursortasten. Betätigen Sie dann jeweils die Taste „D“.



7. Tragen Sie den C-Quellcode zum Ein- und Ausschalten der LED in die DO-Elemente ein. Selektieren Sie dazu das betreffende DO-Element und wählen ENTER oder Doppelklick. Mit ESC verlassen Sie den Editiermodus.

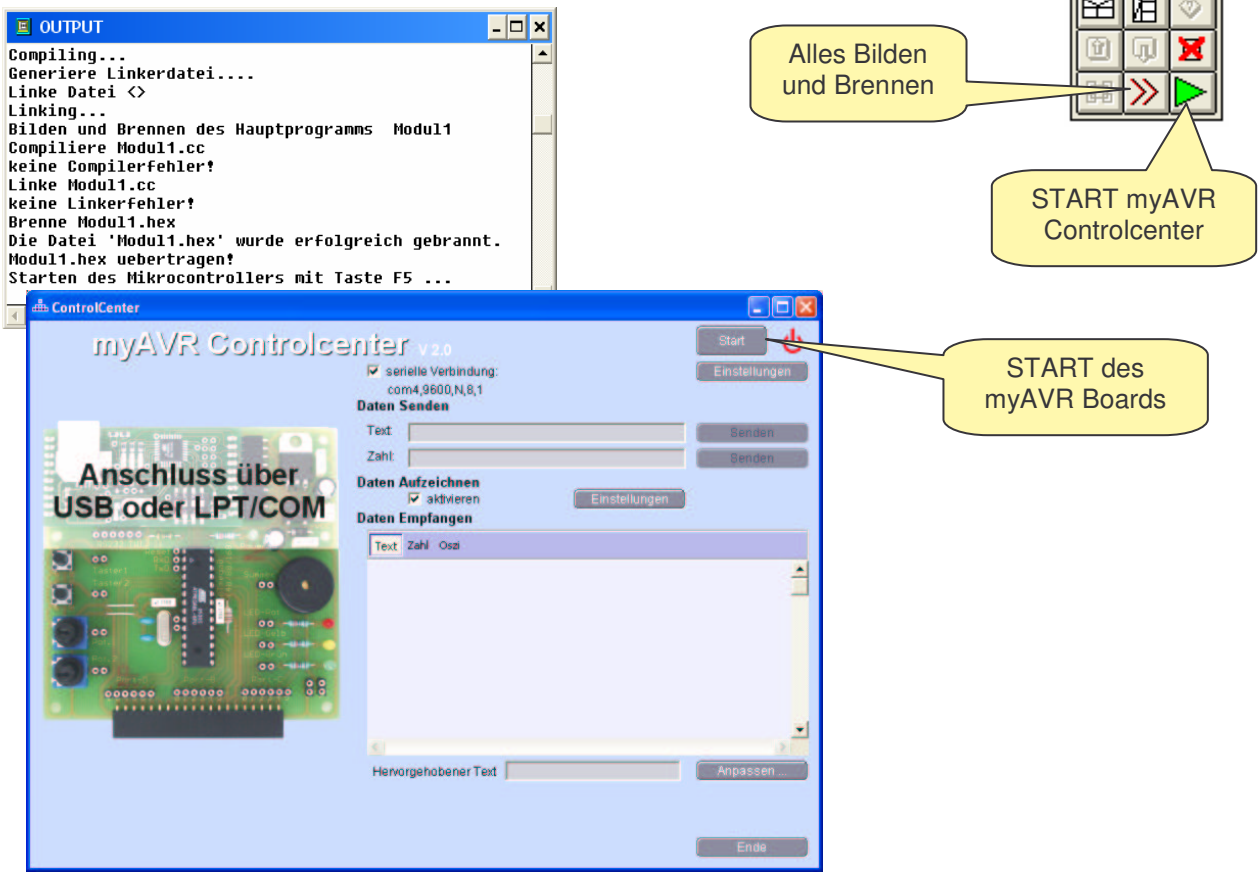




8. Generieren Sie den Quellcode für dieses Struktogramm. Betätigen Sie die Taste F8 und danach die Taste F3.

10.5 Übersetzen, Brennen und Testen

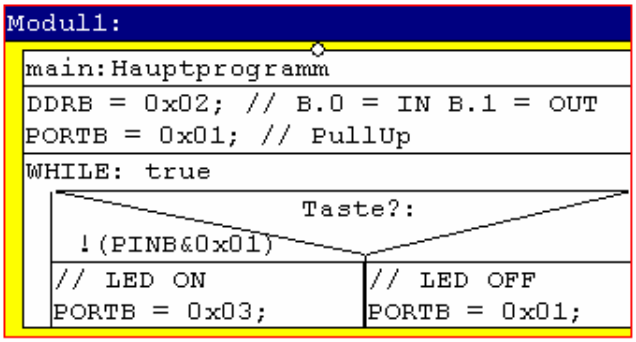
Der so generierte Quellcode kann jetzt kompiliert, gelinkt und gebrannt werden. Wenn die HEX-Datei erfolgreich übertragen wurde, kann zum Testen das myAVR Controlcenter gestartet werden. Nutzen Sie dazu die Toolbox „Objekt“.



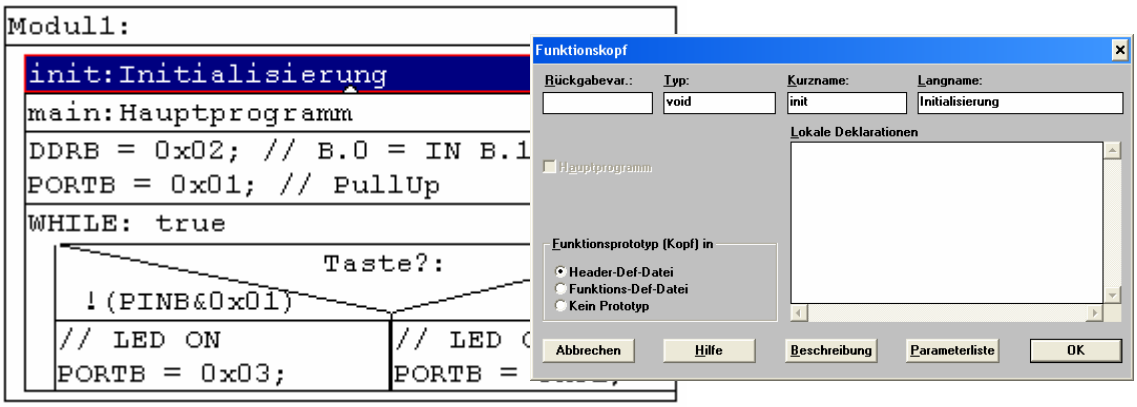
10.6 Funktionen (Unterprogramme) im Struktogramm

Funktionen sind in der Sprache C das wichtigste Instrument zur übersichtlichen und wartungsfreundlichen Gestaltung von Programmen. Es gibt Standardfunktionen wie **`itoa`** oder **`sprintf`** die durch die Verwendung von Bibliotheken wichtige, immer wieder benötigte Algorithmen zur Verfügung stellen. Für den C-Programmierer ist die Entwicklung eigener Funktionen genauso wichtig wie die Verwendung von Bibliotheksfunktionen. Um eine Funktion im Struktogramm zu entwerfen, gehen Sie wie folgt vor (Erweiterung des Beispiels durch die Funktion **`void init()`**, Auslagern der Initialisierung):

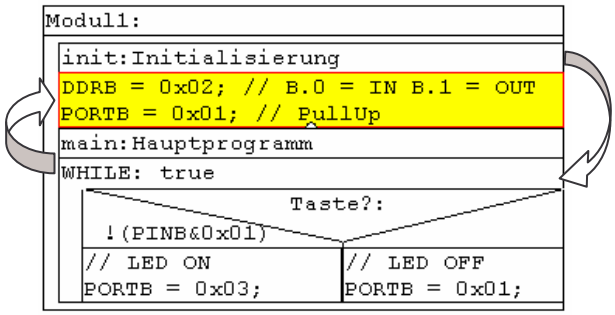
1. Selektieren Sie das Modul mit der Maus oder den Cursortasten.

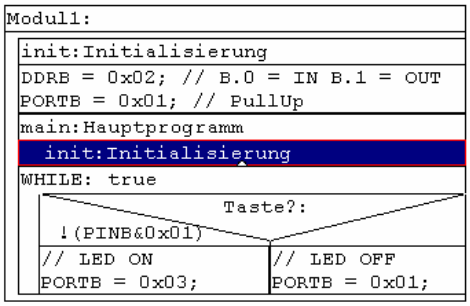


2. Fügen Sie durch Betätigen der Taste „B“ (Begin / Funktion) eine neue Funktion ein. Mit ENTER oder Doppelklick auf der Funktion können Sie Name, Typ und Parameter bearbeiten.



3. Verschieben Sie per Drag & Drop die Aktion „Initialisierung“ in die Funktion **`init`**. Den Funktionsaufruf erzeugen Sie ebenfalls per Drag & Drop. Selektieren Sie den Funktionskopf und ziehen Sie diesen an die gewünschte Aufrufstelle.



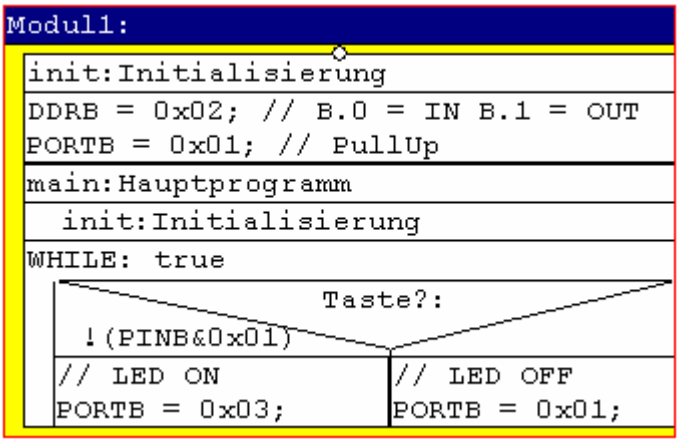


Mit der Tastenfolge F8 und F3 können Sie sich das Ergebnis der Umstrukturierung im Quellcode ansehen.

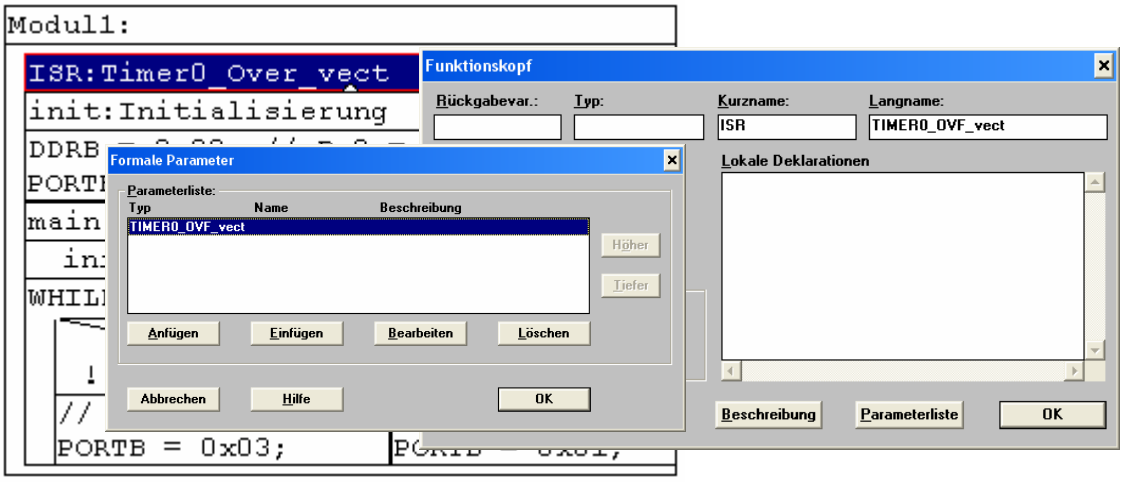
10.7 Interrupt-Service-Routinen (ISR) im Struktogramm

Interrupt-Service-Routinen (im Weiteren ISR) werden in AVR C durch das Schlüsselwort ISR gekennzeichnet. Sie bilden eigenständige Funktionen. Die Besonderheit bei der Realisierung einer ISR liegt darin, dass es sich hier um ein C-Makro handelt und nicht um eine echte Funktion. Um eine ISR im Struktogramm zu entwerfen, gehen Sie wie folgt vor (Erweiterung des Beispiels um die **`ISR Timer0 Overflow`**):

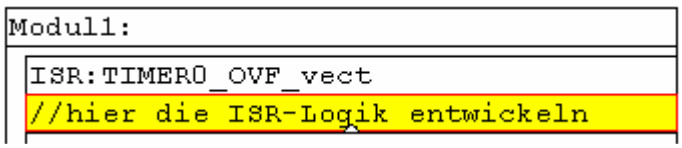
1. Selektieren Sie das Modul mit der Maus oder den Cursortasten.



2. Fügen Sie mit der Taste „B“ eine neue Funktion ein. Per Doppelklick oder Taste ENTER können Sie Name, Typ und Parameter definieren (Typ = Leerzeichen, Kurzname = ISR, Parametertyp = TIMER0_OVF_vect, Parametername = Leerzeichen).



3. Fügen Sie entsprechende Elemente (Do, Alternative, While usw.) für die ISR-Logik ein.



11 Entwicklung eines Klassendiagramms mit SiSy AVR

11.1 Einleitung

Mit objektorientierten Programmiersprachen hat der Entwickler mächtige Sprachmittel, um komplexe Systeme realisieren zu können. C++ ist eine weit verbreitete objektorientierte Standardsprache. Als Visualisierungsmittel objektorientierter Programme gilt die international standardisierte Beschreibungssprache UML (Unified Modeling Language). SiSy AVR bietet dem Entwickler das UML Klassendiagramm mit Codegenerierung für AVR C++. Der folgende Abschnitt beschreibt die Handhabung des Klassendiagramms in SiSy AVR. Die Abbildung 33 zeigt Ihnen eine Kurzübersicht der Modellierungselemente des UML Klassendiagramms.

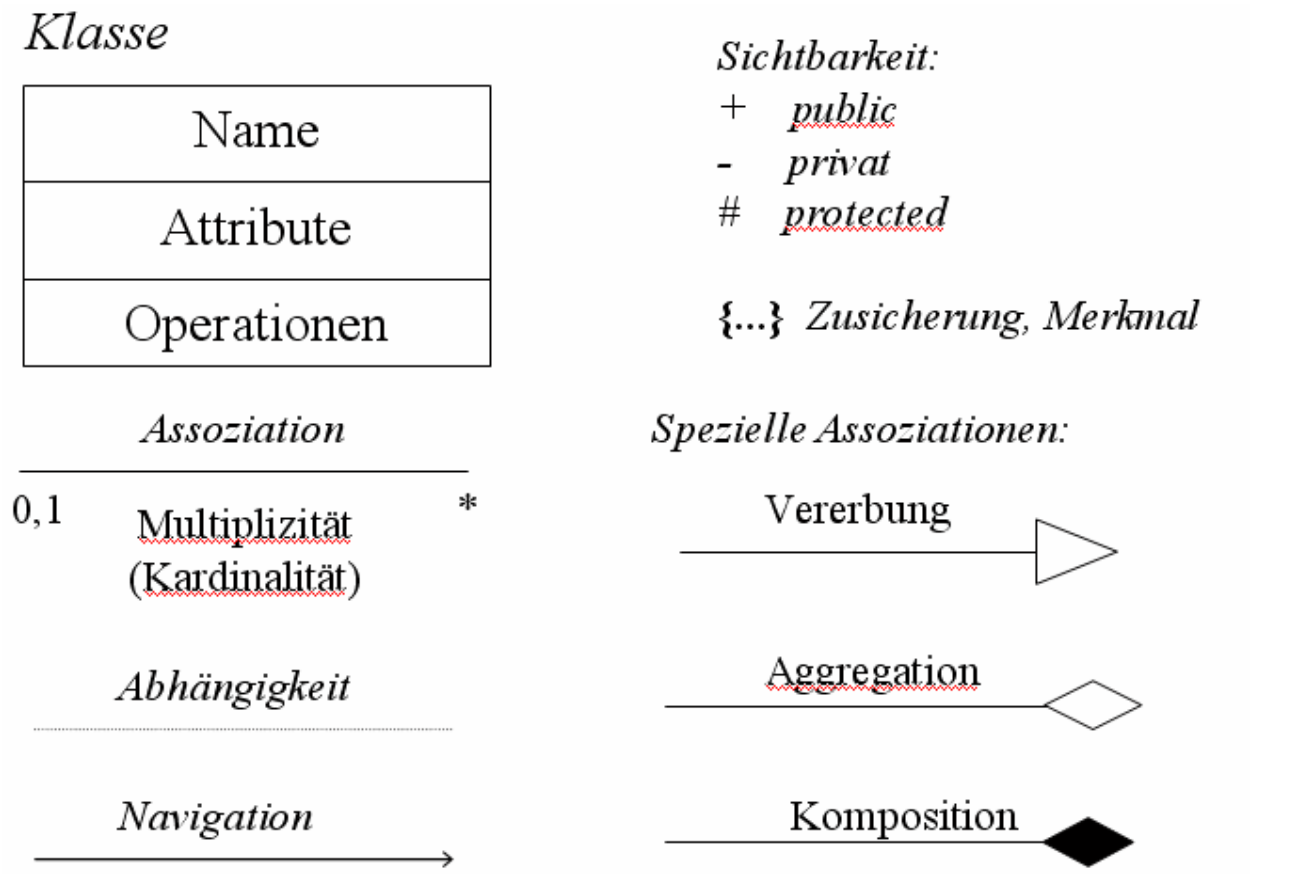


Abbildung 33: Kurzübersicht Elemente des UML Klassendiagramms

Schreibweise von Attributen:
Attribute beginnen mit einem Kleinbuchstaben.

```
Sichtbarkeit name : Typ = Initialwert {Merkmal}{Zusicherung}

# temperatur : uint8_t = 25  {temeratur > 0}
```

Schreibweise von Operationen:
Operationen beginnen mit einem Kleinbuchstaben.

```
Sichtbarkeit name (Parameter:Typ = Standardwert ...) : Rückgabetyp {Merkmal}{Zusicherung}

+ setTemperatur ( t : integer = 25 ) : BOOL {t > 0}
```

11.2 Aufgabenstellung

Um die Arbeitsweise und den Umgang mit dem Klassendiagramm in SiSy zu erlernen, realisieren Sie bitte das folgende einfache Beispiel.

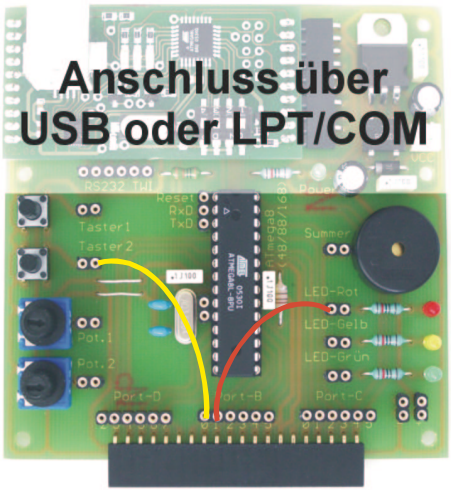


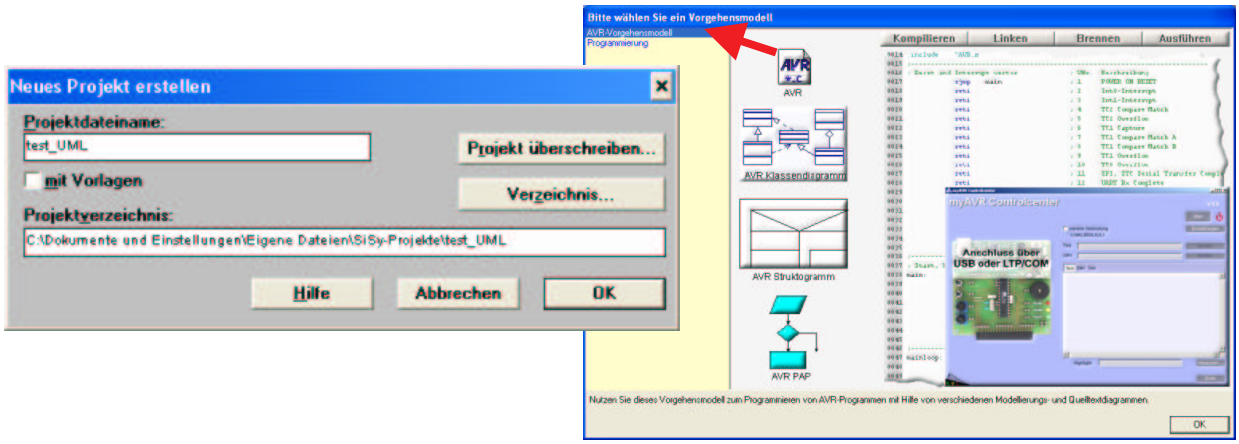
Abbildung 34: UML Beispiel

Es soll eine Mikrocontrollerlösung entwickelt werden, bei der auf Tastendruck eine LED eingeschaltet wird. Dabei soll der Taster an Port B 0 und die LED an Port B 1 angeschlossen werden

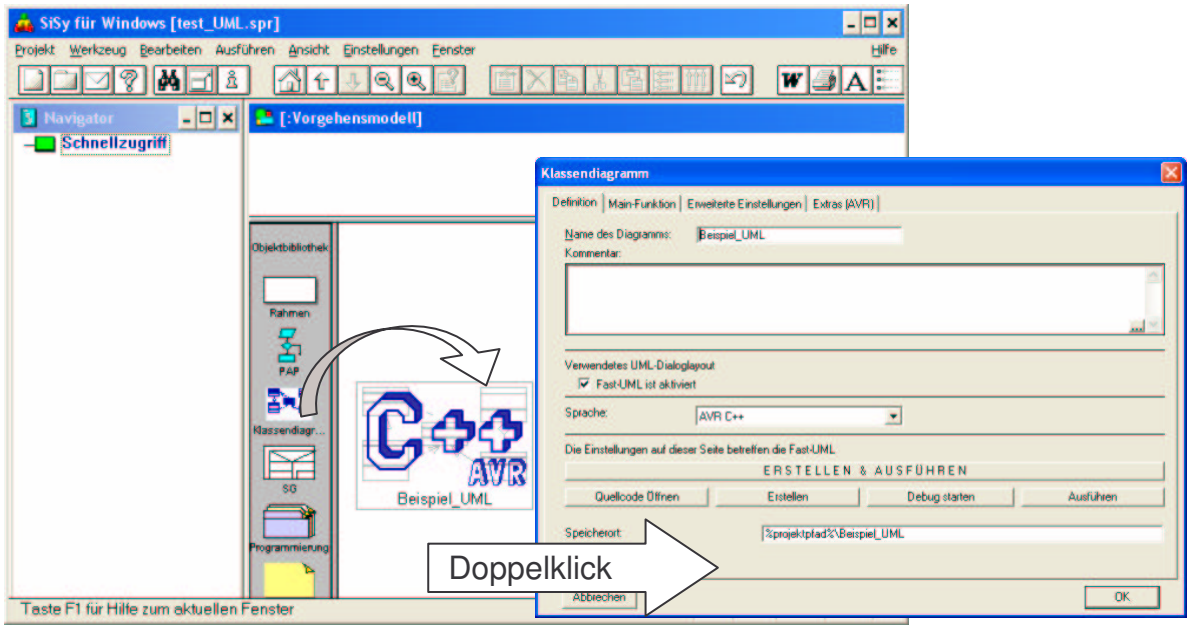
11.3 Vorbereitung

Starten Sie ggf. SiSy.

Erstellen Sie ein neues SiSy-Projekt (Hauptmenu -> Projekt -> Neu) mit dem Namen „Test_UML“. Wählen Sie das Vorgehensmodell „AVR-Vorgehensmodell“.

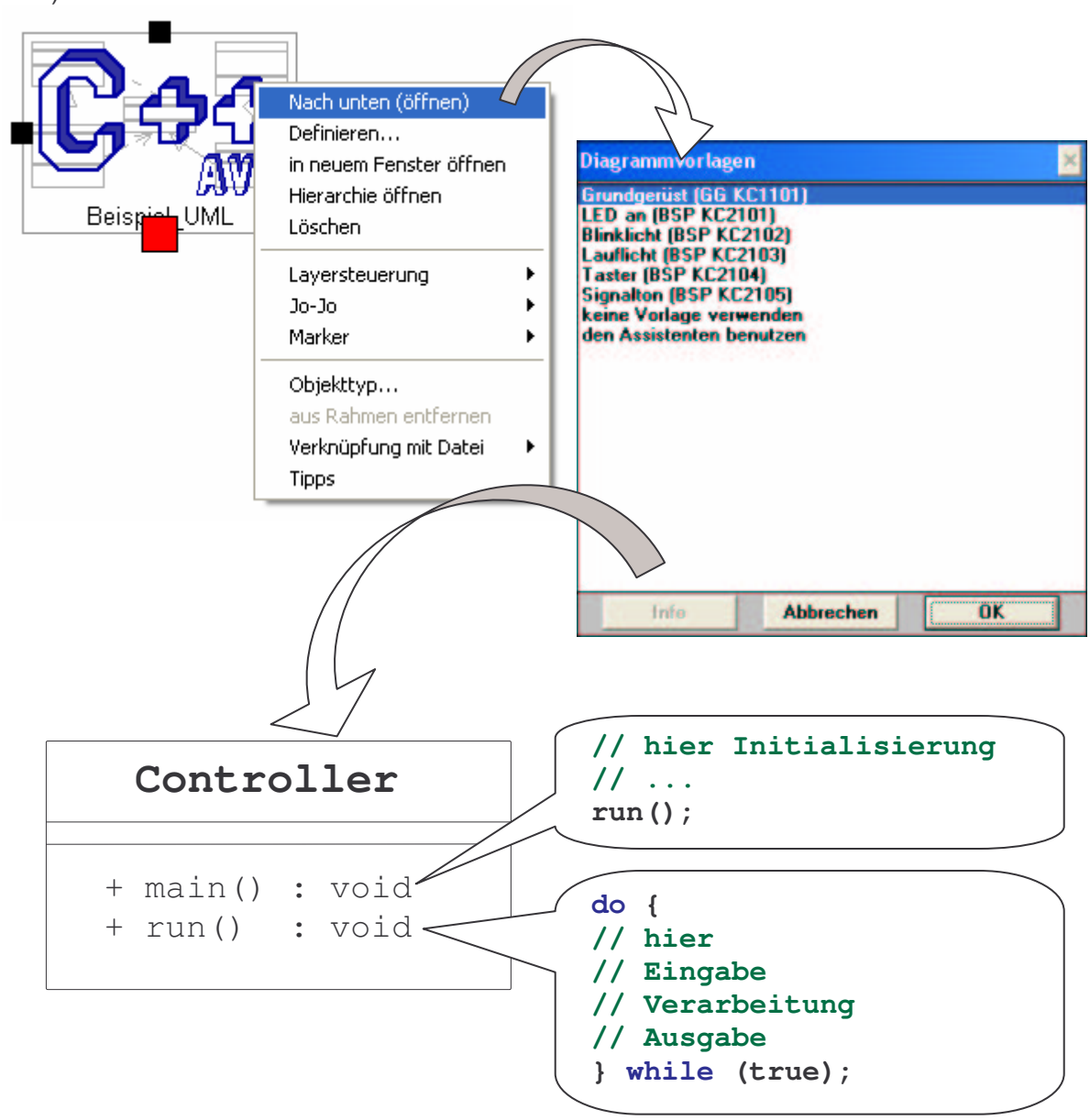


Legen Sie im weiteren Verlauf die AVR-Grundeinstellungen für das Projekt fest. Ziehen Sie ein Objekt vom Typ „Klassendiagramm“ per Drag & Drop in das Diagrammfenster. Legen Sie den Namen für das Klassendiagramm fest.



11.4 Grundstruktur laden

Um das Klassenmodell zu entwickeln, muss das Klassendiagramm geöffnet werden. Wählen Sie dazu auf dem Objekt rechte Maustaste -> Kontextmenü -> nach unten (öffnen).



Die Struktur einer objektorientierten Mikrocontrollerlösung in SiSy erfordert im Klassenmodell eine Applikationsklasse (Hauptklasse), die sich dadurch auszeichnet, dass diese über eine Methode (Operation) mit dem Namen **main** verfügt. Der Codegenerator erzeugt eine Instanz diese Klasse und ruft die Main-Methode auf.

```
#define cpp_Test_UML
#define F_CPU 3686400
#include <io.h>
#include "Controller.cc"

main (void)
{
    Controller MyApp;
    MyApp.main();
}
```

11.5 Systemstruktur entwerfen

Die Systemstruktur einer objektorientierten Anwendung bildet die Objekte und deren Beziehungen im Programm ab, welche im realen System als physische Elemente vorhanden sind. Als Bauplan der Objekte dienen Klassendeklarationen, welche die Eigenschaften (Attribute) und das Verhalten (Methoden/Operationen) der Objekte beschreiben. Das Klassendiagramm beschreibt also die Struktur der Klassen (Baupläne der Objekte) und die Beziehungen zwischen den Klassen. In unserer Aufgabenstellung finden wir die Objekte des Systems als **Substantive,** deren Beziehungen und Verhalten als **Verbalphrasen**.

*Es soll eine Mikro**controller**lösung entwickelt werden, bei der durch **drücken** eines **Tasters** eine **LED eingeschaltet** wird. Dabei soll der Taster an Port B 0 und die LED an Port B 1 **angeschlossen** werden.*

Daraus lässt sich folgende Klassenstruktur ableiten:

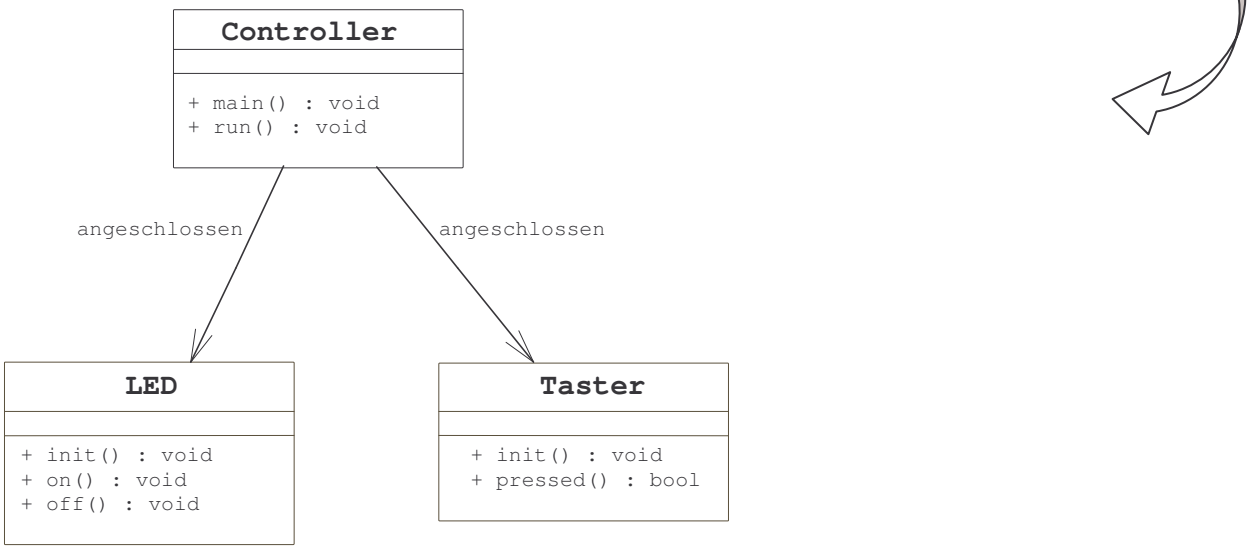
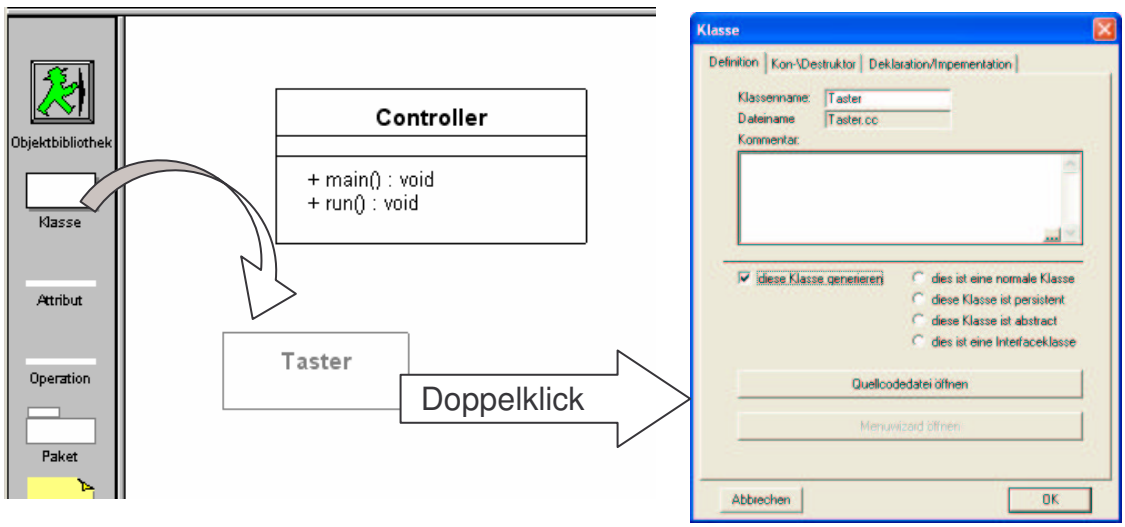


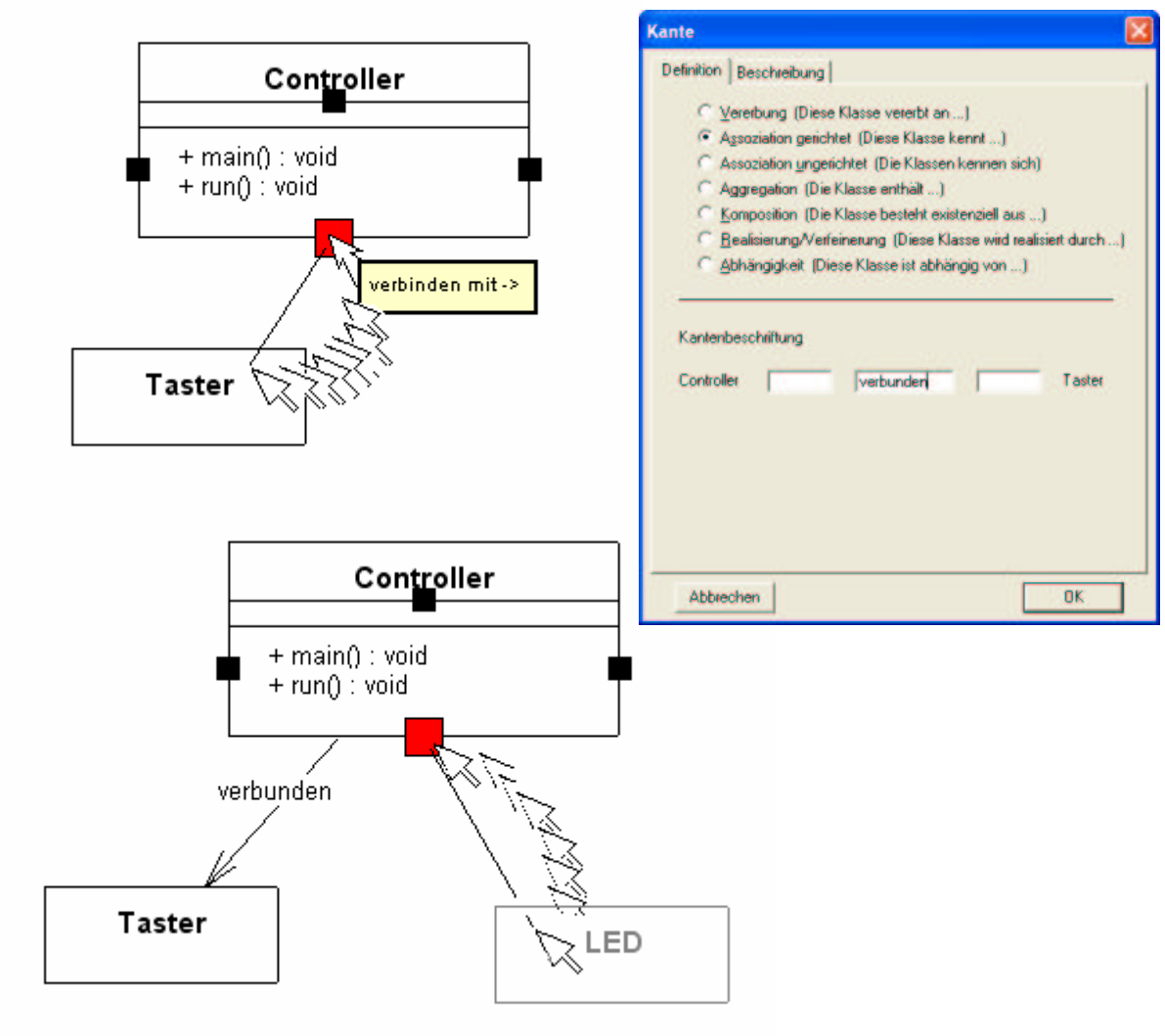
Abbildung 35: Klassenstruktur, Systementwurf mit dem UML Klassendiagramm

Zum Erstellen dieses Klassenmodells sind folgende Arbeitsschritte nötig:

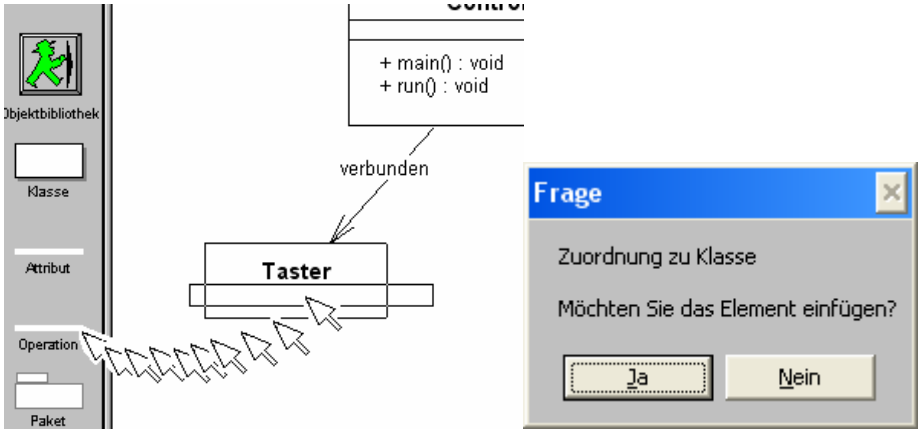
1. **Klassen einfügen und definieren**, ziehen Sie dazu das Objekt vom Typ „Klasse“ per Drag & Drop aus der Objektbibliothek in das Diagramm. Definieren Sie den Namen der Klasse und setzen die Option „diese Klasse generieren“.



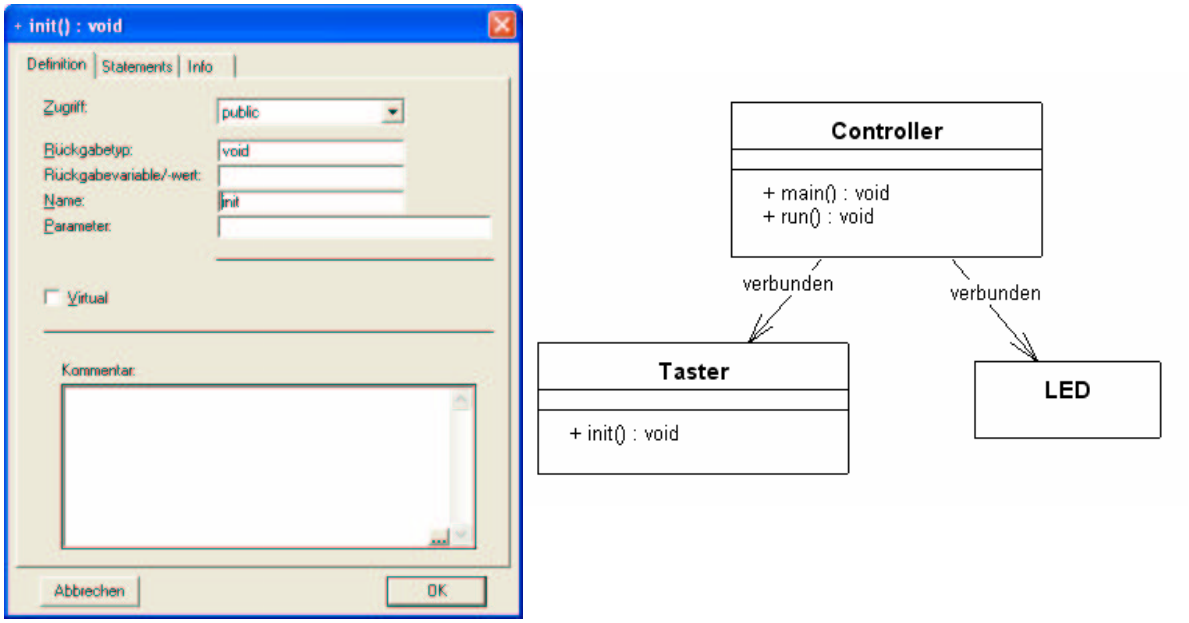
2. **Klassen verbinden**, selektieren Sie die Klasse, von der aus eine Verbindung gezogen werden soll. Ziehen Sie per Drag & Drop ausgehend vom roten Verteiler eine Verbindung auf die gewünschte Klasse. Wählen Sie den Verbindungstyp, zum Beispiel „Assoziation gerichtet“ und beschriften Sie die Verbindung.



3. **Operationen und Attribute einfügen und definieren**, ziehen Sie dazu ein Objekt vom Typ „Operation“ oder „Attribut“ aus der Objektbibliothek auf die Klasse, in die das Attribut oder die Operation eingefügt werden soll. Bestätigen Sie die Sicherheitsabfrage zum Einfügen des Elementes.



Definieren Sie Zugriff (Sichtbarkeit), Name, Typ und Parameter der Operation bzw. Zugriff (Sichtbarkeit), Name, Typ und Initialwert des Attributes.



Vervollständigen Sie das Klassenmodell entsprechend der Abbildung 36.

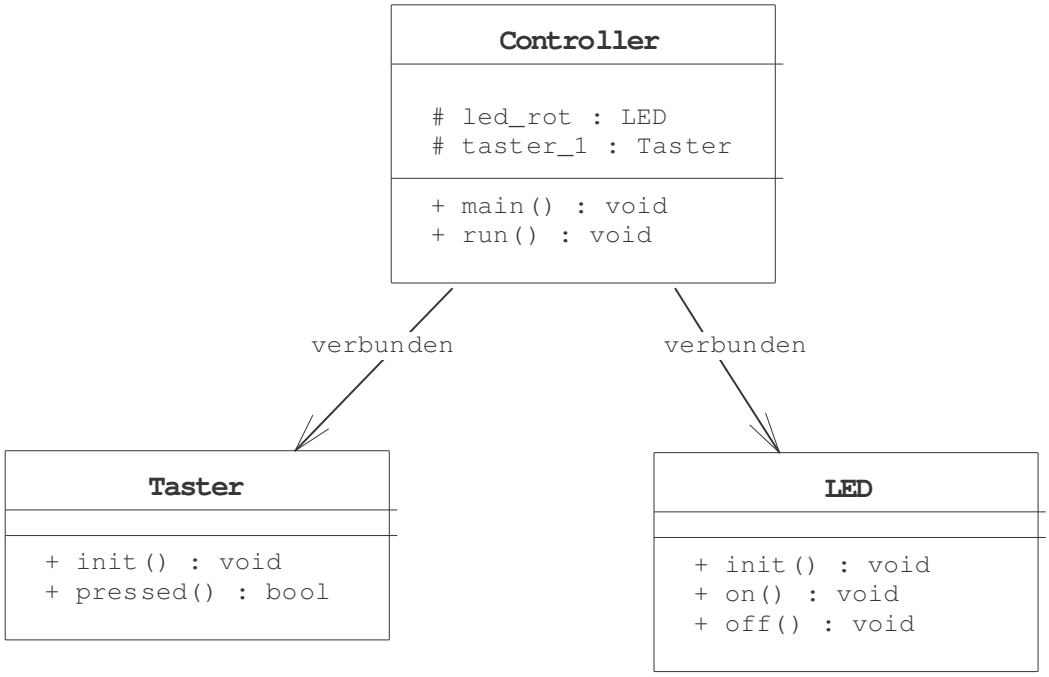
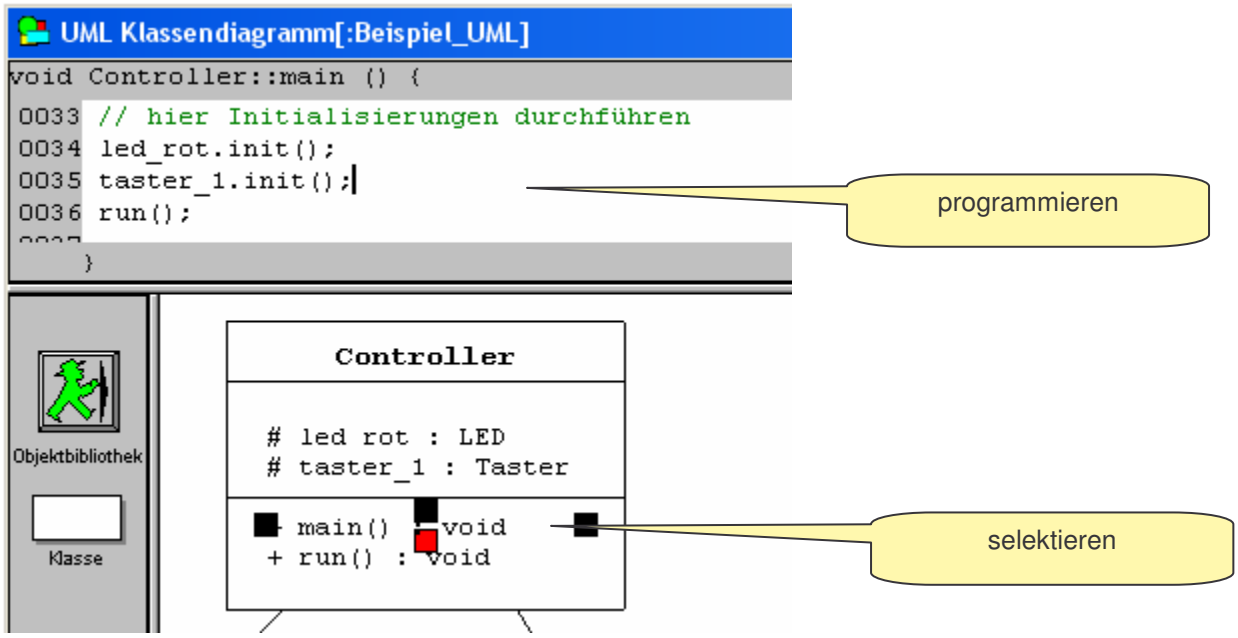
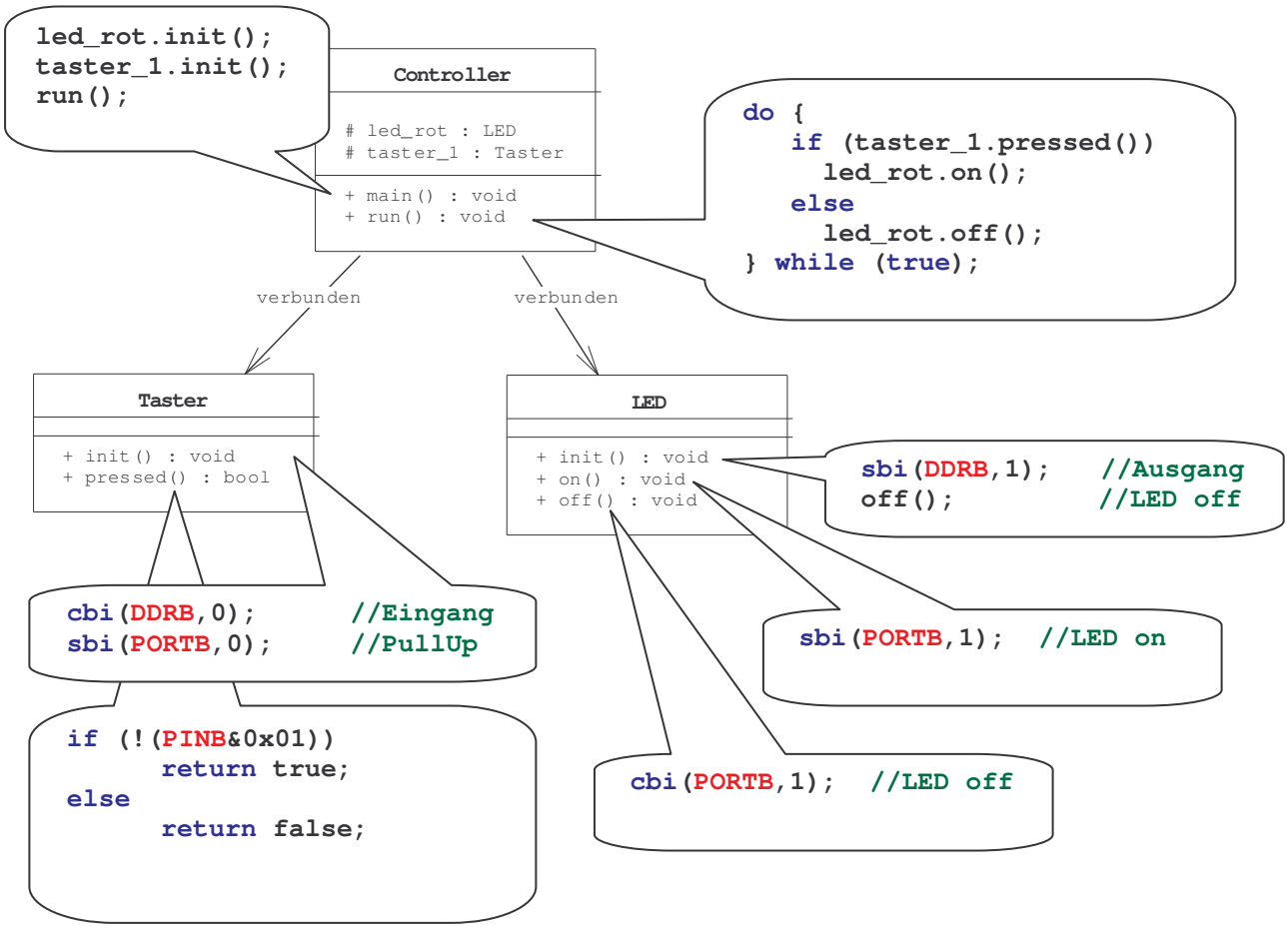


Abbildung 36: endgültiger Systementwurf mit dem UML Klassendiagramm

11.6 Systemverhalten programmieren

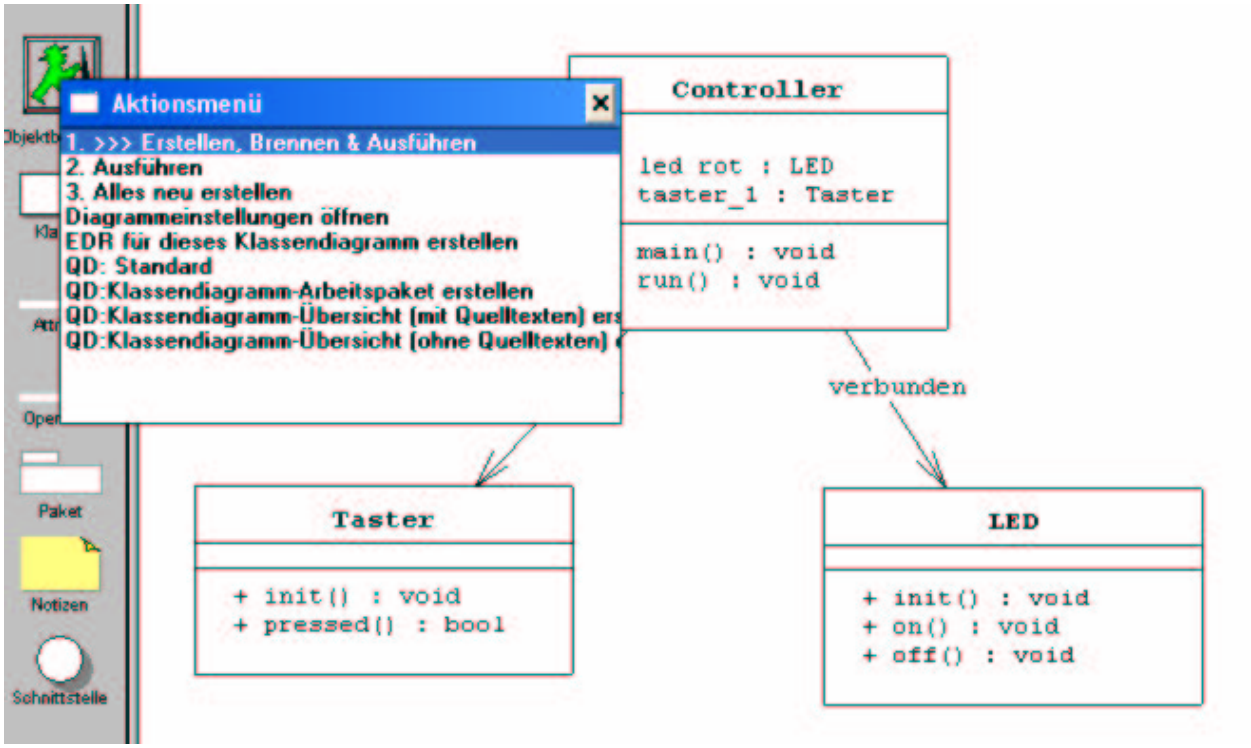
Die Operationen müssen mit der entsprechenden Logik versehen werden. Sie können den Quellcode der Operationen bearbeiten, indem Sie die gewünschte Operation selektieren und im Beschreibungs- / Quellcodefester die Befehle eingeben.





11.7 Übersetzen, Brennen und Testen

Die Codegenerierung aus dem UML Klassendiagramm, das Kompilieren, Linken und Brennen kann über das Ampelmännchen gestartet werden.



Sie erhalten im Ausgabefenster ein Protokoll der ausgeführten Aktionen.

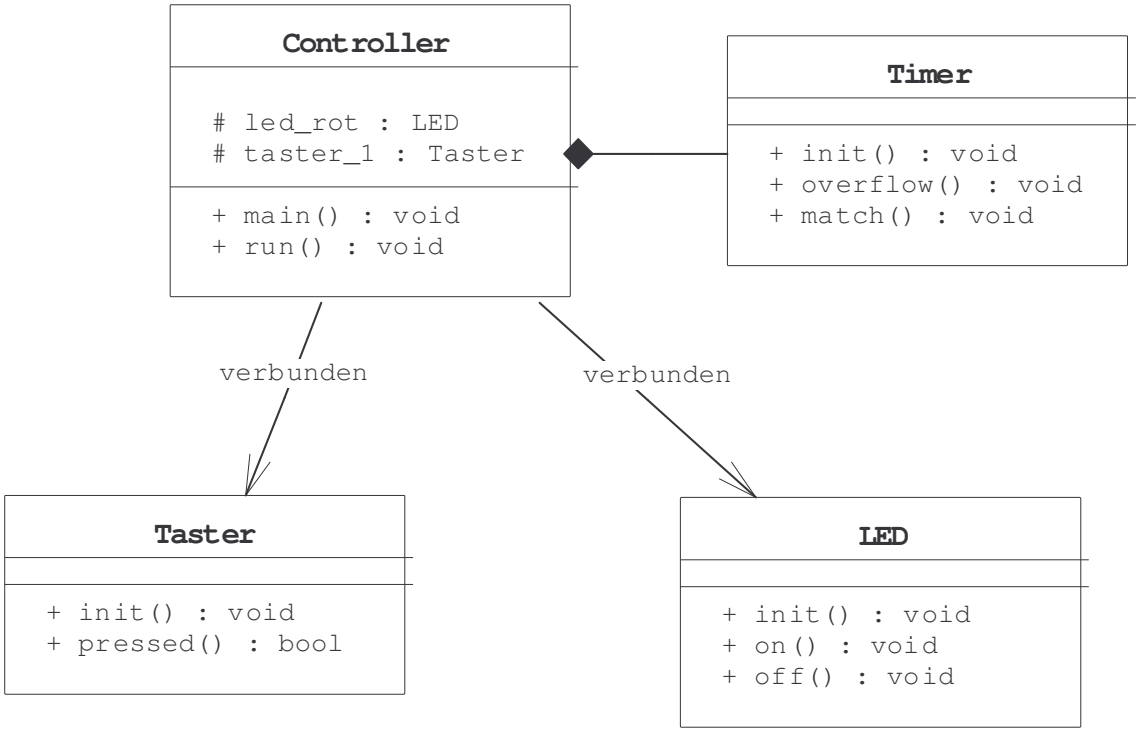
Generiere Quelltexte...
Suche Diagramme: ok
Lösche Dateien in C:\DOKUME~1\...\TEST_UML\Beispiel_UML.
Prüfe Diagramme: 1 von 1 Paketen
Bearbeite Diagramme...
Speichere Beispiel_UML.cc.
Speichere Controller.cc.
Speichere Controller.h.
Speichere Taster.cc.
Speichere Taster.h.
Speichere LED.cc.
Speichere LED.h.
Kompiliere die Datei Beispiel_UML.cc.
Linke Datei Beispiel_UML.hex.

Brenne Daten aus Datei Beispiel_UML.hex.
Das AVR Gerät ist initalisiert und bereit.
Geräte Signatur: = 0x07931e
Lösche die Daten auf dem Flash.
Einlesen der Datei: "Beispiel_UML.hex"
Schreibe die Daten auf den Flash. (1078 bytes):
1078 Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Vergleiche ...
1078 Bytes erfolgreich verglichen, keine Fehler.

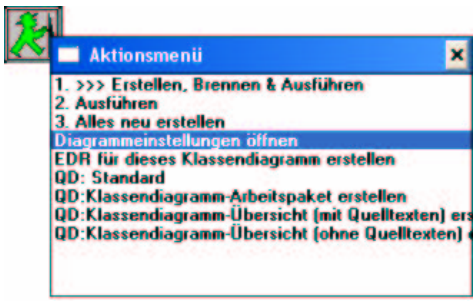
11.8 Interrupt-Service-Routinen (ISR) im Klassendiagramm

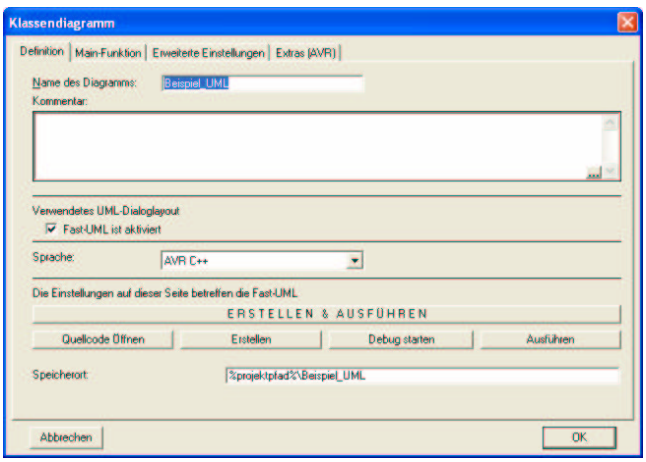
Interrupt-Service-Routinen (im weiteren ISR) werden in AVR C++ durch das Schlüsselwort ISR gekennzeichnet. Sie bilden eigenständige Funktionen. Die Besonderheit bei der Realisierung einer ISR liegt darin, dass es sich hier um ein C-Makro handelt und nicht um eine echte Funktion. Diese können also keine Methode einer Klasse sein. Um eine ISR im Klassendiagramm zu realisieren gehen Sie wie folgt vor (Erweiterung des Beispiels um die Klasser Timer und der **`ISR Timer0 Overflow`**):

1. Die Klasse für die interruptfähige Komponente modellieren und die Methode einfügen, welche beim Interrupt ausgeführt werden soll. In unserem Beispiel ist es die Klasse **`Timer`** mit der Methode **`overflow`**. Die Initialisierung des Timers muss entsprechend erfolgen.

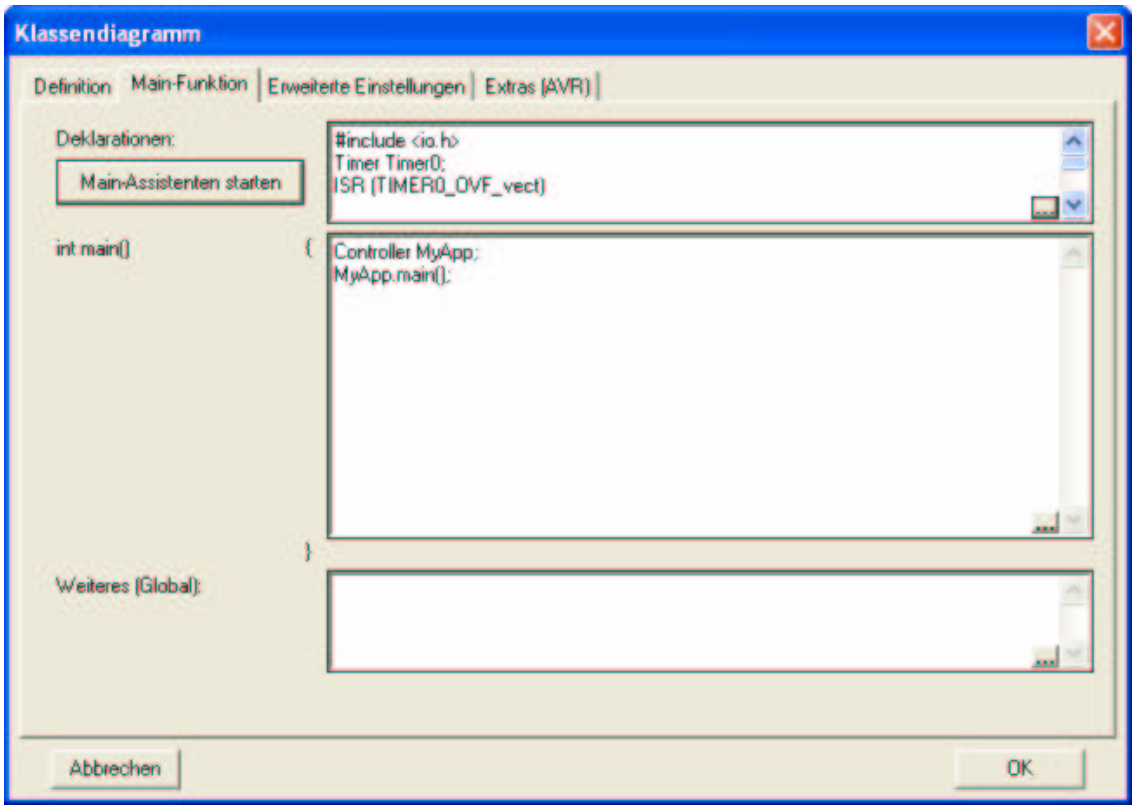


2. Den Dialog „Diagrammeinstellungen“ öffnen. Sie erreichen diesen Dialog über das Ampelmännchen, Menüpunkt: „Diagrammeinstellungen öffnen“





3. Wählen Sie in diesem Dialog das Dialogfeld „Main-Funktion“ und ergänzen im Eingabefeld „Deklarationen“ das ISR-Makro.



Beispiel:

```
#include  <io.h>

// Instanz der interruptfähigen Klasse anlegen
Timer Timer0;

// Interuptmakro mit aufruf der betreffenden Methode
ISR (TIMER0_OVF_vect)
{
     Timer0.overflow();
}
```

In der Interrupt-Service-Routine rufen Sie einfach die Methode der betreffenden Klasse auf.

12 Einstellungen Fuse- und Lock-Bits mit SiSy

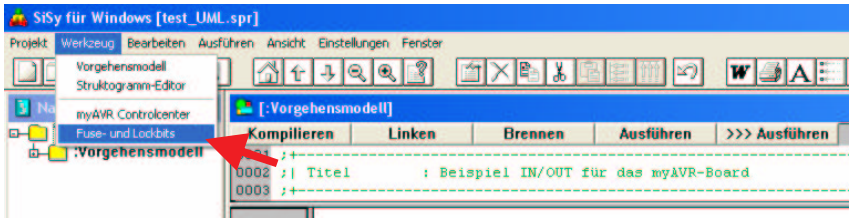
12.1 Einleitung

Fuse- und Lockbits (engl. fuse = Sicherung, engl. lock = Schloss) nennt man die Bits in bestimmten Registern des AVR zum Konfigurieren des Controllers. Die Fusebits müssen über ein entsprechendes Interface (Software) eingestellt werden. Der normale Programmiermodus verändert die Fuse- und Lockbits nicht. Je nach Controllertyp sind unterschiedliche Fuse- und Lockbits verfügbar. Die verbindliche und exakte Beschreibung findet man im jeweiligen Datenblatt des Controllers. Das falsche Setzen der Fuse- und Lockbits zählt zu den häufigsten Problemen bei der Programmierung von AVR-Controllern, daher sollte hier mit Umsicht vorgegangen werden. Das Verändern der Fusebits sollte man nicht als Anfänger vornehmen. Das geöffnete Datenblatt zur Überprüfung der Konfiguration ist das wichtigste Instrument um Fehler zu vermeiden.

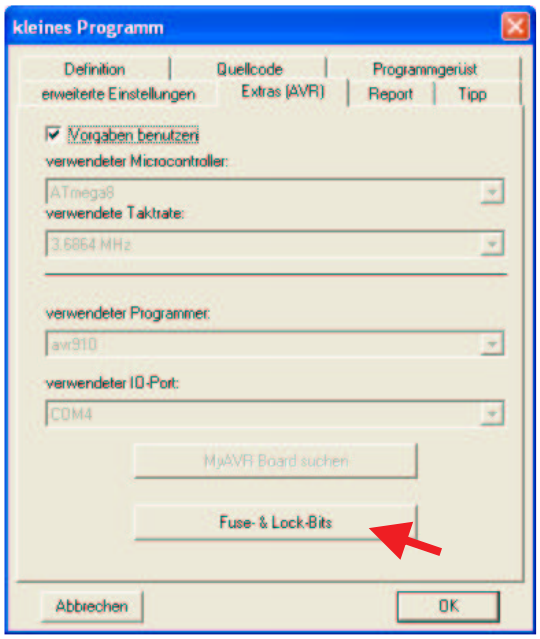
12.2 Fuse- und Lockbits, Benutzeroberfläche in SiSy AVR

Sie erreichen die Benutzeroberfläche zum Auslesen, Verändern und Programmieren der Fuse- und Lockbits in SiSy AVR wie folgt:

1. Hauptmenü -> Werkzeuge -> Fuse- und Lockbits



2. Auf Programm-Objekten wie „kleines Programm“, „Programm“, „PAP“ usw. rechte Maustaste -> Kontextmenü -> Definieren … Dialogfeld Extras (AVR) -> Schaltfläche „Fuse- & Lock-Bits“.



Beim Start der Fuse- und Lockbits-Benutzeroberfläche wird eine Verbindung zum Controller aufgebaut, der Controllertyp ermittelt, die Fuse- und Lockbit-Definitonen des Controllers geladen, die Fuse- und Lockbit Einstellungen des Controllers ausgelesen und angezeigt. Dieser Vorgang kann je nach Controllertyp und Verbindung einige Sekunden

dauern. Die Verbindung zum Controller wird, solange diese Benutzeroberfläche offen ist, dauerhaft offen gehalten.

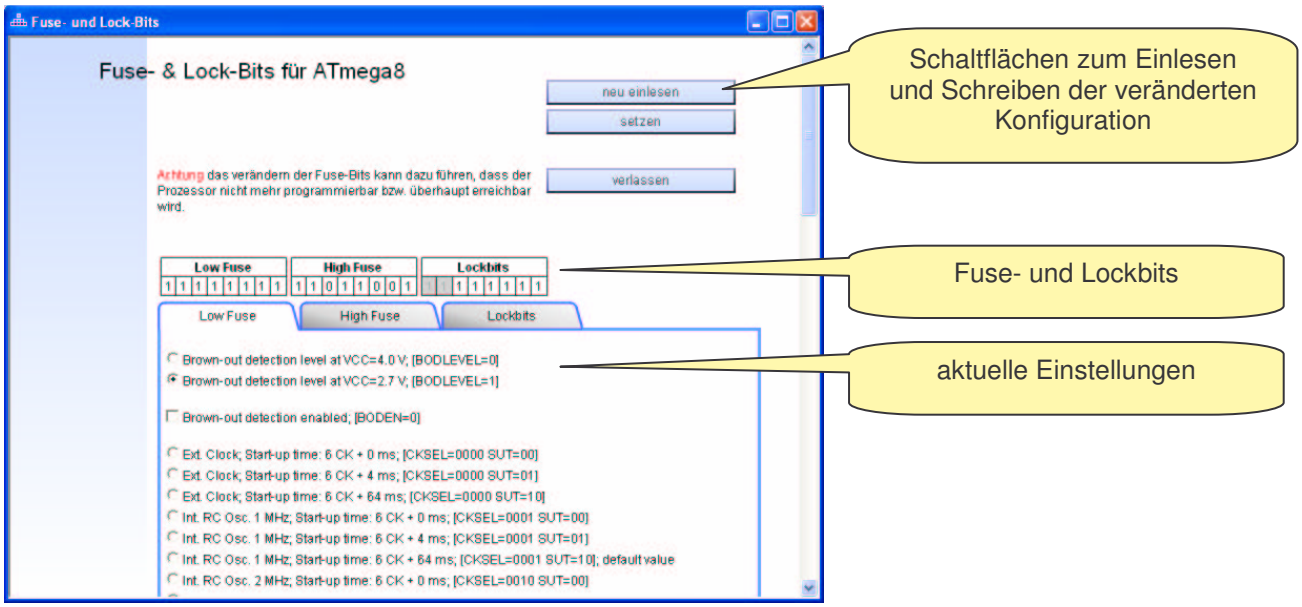

Fuse- und Lock-Bits
Fuse- & Lock-Bits für ATmega8
neu einlesen
setzen
verlassen
Achtung das verändern der Fuse-Bits kann dazu führen, dass der Prozessor nicht mehr programmierbar bzw. überhaupt erreichbar wird.
Low Fuse
High Fuse
Lockbits
Schaltflächen zum Einlesen und Schreiben der veränderten Konfiguration
Fuse- und Lockbits
aktuelle Einstellungen

Abbildung 37: Fuse- und Lockbits des ATmega8

Die Benutzeroberfläche passt sich dem ermittelten Controller und den dazugehörigen Definitionsdaten automatisch an. Es werden immer nur die Optionen angezeigt, die zum ermittelten Controller gehören. Vergleichen Sie dazu immer das betreffende Datenblatt. Es ist nicht zulässig, den Programmer oder den Controller während der Sitzung zu entfernen oder zu wechseln. Dazu ist dieses Fenster zu schließen, danach der Controller oder Programmer zu wechseln und die Benutzeroberfläche erneut zu starten.

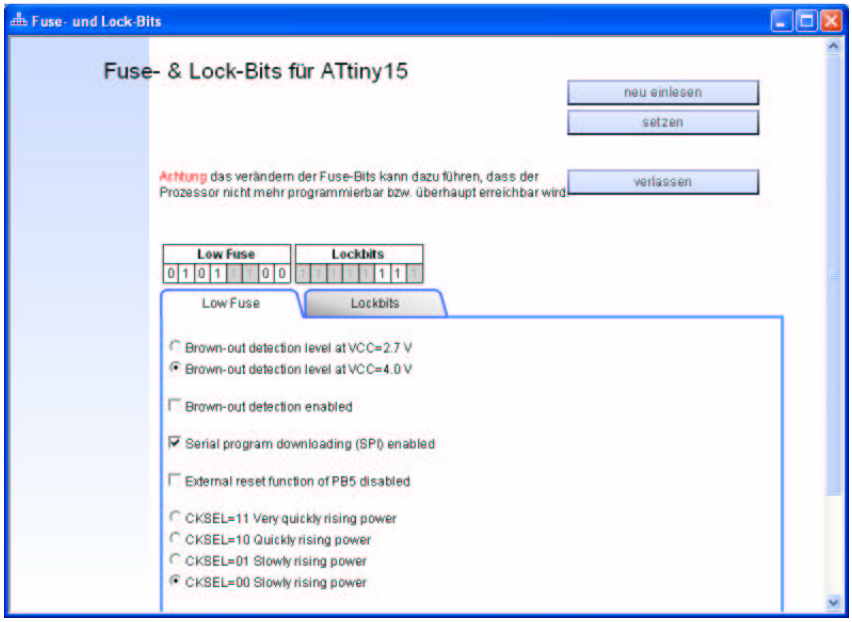

Fuse- und Lock-Bits
Fuse- & Lock-Bits für ATtiny15
neu einlesen
setzen
verlassen
Achtung das verändern der Fuse-Bits kann dazu führen, dass der Prozessor nicht mehr programmierbar bzw. überhaupt erreichbar wird.
Low Fuse
Lockbits

Abbildung 38: Fuse- und Lockbits des ATtiny15

12.3 Fuse- und Lockbits verändern

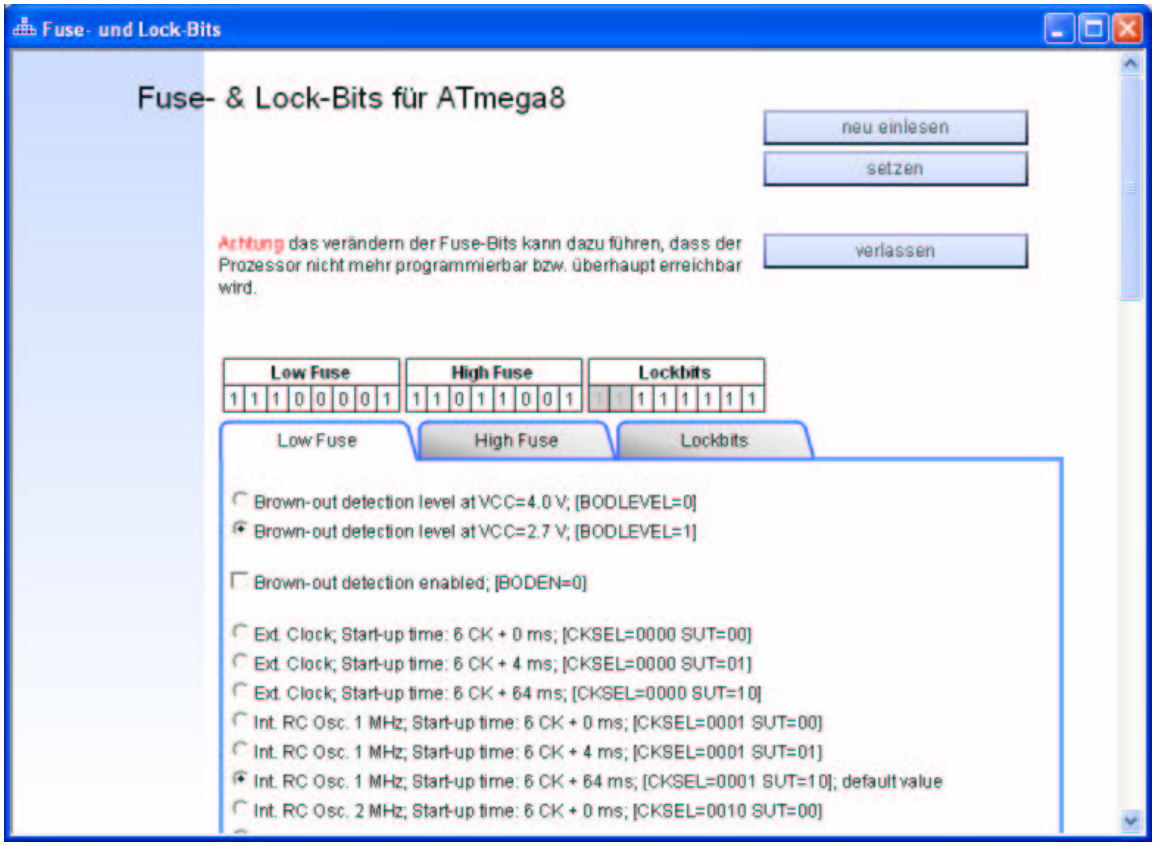
Zum Verändern der Fuse- und Lockbits sollte der entsprechende Abschnitt im Datenblatt des Controllers studiert werden. Über die Dialogfelder Low-, High- und Extended-Fuse- sowie Lockbits können die einzelnen Optionen bequem ausgewählt werden. Die Änderungen werden im Anzeigebereich für die Fuse- und Lockbits visualisiert. Erst mit dem Betätigen der Schaltfläche „setzen“ werden die neuen Einstellungen an den Controller übertragen.

Beachte: Falsche Einstellungen der Fuse- oder Lockbits können dazu führen, dass der Controller in der aktuellen Hardware nicht mehr angesprochen werden kann.
Häufige Fehleinstellungen durch unerfahrene Entwickler sind:

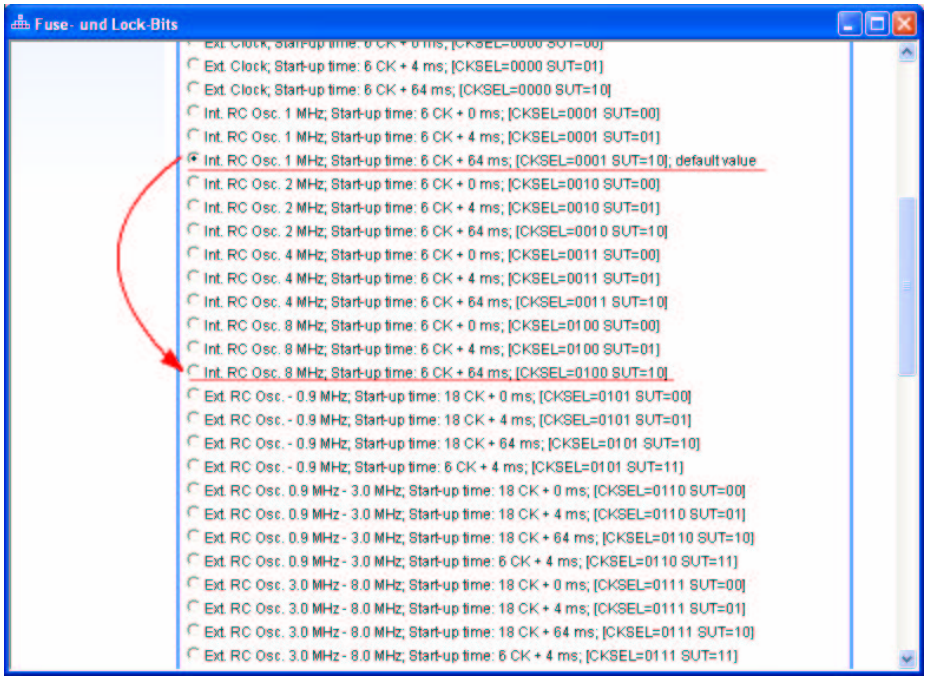
- **Reset disable ->** führt dazu, dass kein ISP mehr möglich ist
- **ISP enable ausgeschaltet ->** führt dazu, dass kein ISP mehr möglich ist
- **Taktquelle umgeschaltet ->** führt u.U. dazu, dass der Controller nicht arbeitet

Im Folgenden wird die Vorgehensweise beschrieben, wie die Taktquelle eines ATmega8 vom internen 1 MHz Oszillator auf intern 8 MHz umgeschaltet wird.

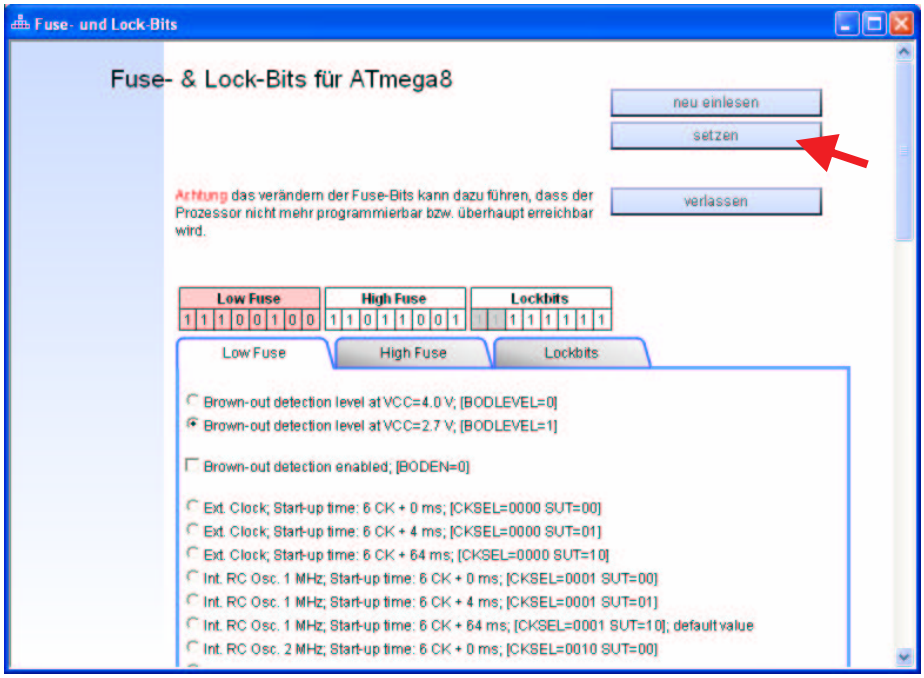
1. Das Board und den Programmer anschießen.
2. Benutzeroberfläche für das Verändern der Fuse- & Lockbits starten.


Fuse- und Lock-Bits
Fuse- & Lock-Bits für ATmega8
neu einlesen
setzen
verlassen
Achtung das verändern der Fuse-Bits kann dazu führen, dass der Prozessor nicht mehr programmierbar bzw. überhaupt erreichbar wird.
Low Fuse
High Fuse
Lockbits
1 1 1 0 0 0 0 1
1 1 0 1 1 0 0 1
1 1 1 1 1 1 1 1
Low Fuse
High Fuse
Lockbits
Brown-out detection level at VCC=4.0 V; [BODLEVEL=0]
Brown-out detection level at VCC=2.7 V; [BODLEVEL=1]
Brown-out detection enabled; [BODEN=0]
Ext. Clock; Start-up time: 6 CK + 0 ms; [CKSEL=0000 SUT=00]
Ext. Clock; Start-up time: 6 CK + 4 ms; [CKSEL=0000 SUT=01]
Ext. Clock; Start-up time: 6 CK + 64 ms; [CKSEL=0000 SUT=10]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0001 SUT=00]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0001 SUT=01]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0001 SUT=10]; default value
Int. RC Osc. 2 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0010 SUT=00]

3. in der Liste der Optionen nach unten scrollen und **`Int. RC. Osc. 8MHz`** auswählen. Vergleichen Sie dazu die Beschreibung im Datenblatt des ATmega8.


Fuse- und Lock-Bits
Ext. Clock; Start-up time: 6 CK + 4 ms; [CKSEL=0000 SUT=01]
Ext. Clock; Start-up time: 6 CK + 64 ms; [CKSEL=0000 SUT=10]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0001 SUT=00]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0001 SUT=01]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0001 SUT=10]; default value
Int. RC Osc. 2 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0010 SUT=00]
Int. RC Osc. 2 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0010 SUT=01]
Int. RC Osc. 2 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0010 SUT=10]
Int. RC Osc. 4 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0011 SUT=00]
Int. RC Osc. 4 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0011 SUT=01]
Int. RC Osc. 4 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0011 SUT=10]
Int. RC Osc. 8 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0100 SUT=00]
Int. RC Osc. 8 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0100 SUT=01]
Int. RC Osc. 8 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0100 SUT=10]
Ext. RC Osc. - 0.9 MHz; Start-up time: 18 CK + 0 ms; [CKSEL=0101 SUT=00]
Ext. RC Osc. - 0.9 MHz; Start-up time: 18 CK + 4 ms; [CKSEL=0101 SUT=01]
Ext. RC Osc. - 0.9 MHz; Start-up time: 18 CK + 64 ms; [CKSEL=0101 SUT=10]
Ext. RC Osc. - 0.9 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0101 SUT=11]
Ext. RC Osc. 0.9 MHz - 3.0 MHz; Start-up time: 18 CK + 0 ms; [CKSEL=0110 SUT=00]
Ext. RC Osc. 0.9 MHz - 3.0 MHz; Start-up time: 18 CK + 4 ms; [CKSEL=0110 SUT=01]
Ext. RC Osc. 0.9 MHz - 3.0 MHz; Start-up time: 18 CK + 64 ms; [CKSEL=0110 SUT=10]
Ext. RC Osc. 0.9 MHz - 3.0 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0110 SUT=11]
Ext. RC Osc. 3.0 MHz - 8.0 MHz; Start-up time: 18 CK + 0 ms; [CKSEL=0111 SUT=00]
Ext. RC Osc. 3.0 MHz - 8.0 MHz; Start-up time: 18 CK + 4 ms; [CKSEL=0111 SUT=01]
Ext. RC Osc. 3.0 MHz - 8.0 MHz; Start-up time: 18 CK + 64 ms; [CKSEL=0111 SUT=10]
Ext. RC Osc. 3.0 MHz - 8.0 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0111 SUT=11]

4. Die Optionsseite wieder zurückscrollen und die veränderten Fusebits überprüfen.


Fuse- und Lock-Bits
Fuse- & Lock-Bits für ATmega8
neu einlesen
setzen
verlassen
Achtung das verändern der Fuse-Bits kann dazu führen, dass der Prozessor nicht mehr programmierbar bzw. überhaupt erreichbar wird.
Low Fuse
High Fuse
Lockbits
Brown-out detection level at VCC=4.0 V; [BODLEVEL=0]
Brown-out detection level at VCC=2.7 V; [BODLEVEL=1]
Brown-out detection enabled; [BODEN=0]
Ext. Clock; Start-up time: 6 CK + 0 ms; [CKSEL=0000 SUT=00]
Ext. Clock; Start-up time: 6 CK + 4 ms; [CKSEL=0000 SUT=01]
Ext. Clock; Start-up time: 6 CK + 64 ms; [CKSEL=0000 SUT=10]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0001 SUT=00]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 4 ms; [CKSEL=0001 SUT=01]
Int. RC Osc. 1 MHz; Start-up time: 6 CK + 64 ms; [CKSEL=0001 SUT=10]; default value
Int. RC Osc. 2 MHz; Start-up time: 6 CK + 0 ms; [CKSEL=0010 SUT=00]

5. Die Schaltfläche „setzen“ wählen und die Sicherheitsabfrage bestätigen. Danach sollten die Einstellungen überprüft werden durch das Betätigen der Schaltfläche „neu einlesen“.

Über die Schaltfläche „verlassen“ kann die Sitzung zum Verändern der Fuse- und Lockbits beendet werden. Die Verbindung zum Controller und Programmer wird dann geschlossen.

Anhang: Tastaturbelegung, allgemein

Die Tastenbelegung ist abhängig vom jeweiligen Diagramm und der verwendeten Ausgabe:

F1	Hilfe wird geöffnet
F2	Zoomen
F3	Objekt wird dem Layer zugefügt
F4	Objekt wird vom Layer entfernt
F5	Farbe bei Rahmen wird geändert; Form am Anfang einer Verbindung ändert sich
F6	Bei Rahmen und Verbindungen ändert sich die Form
F7	Bei Rahmen und Verbindungen ändert sich der Linientyp
F8	Form am Ende einer Verbindung ändert sich
F9	Bewirkt, dass der Mittelpunkt einer Kante auf Null gesetzt wird
ESC	Im Diagramm: Nach oben
	Im Text-Editormodus: Beendet ihn mit Rückspeichern
	Im Dialog: Bricht ihn ab ohne zurückzuspeichern
Tab	In der Reihenfolge, in der die Objekte erstellt wurden, werden sie markiert
Umschalttaste + Enter	Objektbeschreibung wird geöffnet
Leertaste	Objektbeschreibung wird geöffnet
Alt + Enter	Dialog Definieren
Strg + Enter	Diagramm nach unten
Strg + `R	Report für selektiertes Objekt
Strg + `A`	Executebefehl ausführen (nur in bestimmten Ausgaben)
Strg + `D`	Diagrammreport wird aufgerufen
Strg + `I`	Import von Diagrammen
Strg + `T`	Tokensteuerung starten/beenden
Strg + `X`	Export von Diagrammen
Strg + `+`	Selektiertes Objekt wird vergrößert (nur in bestimmten Ausgaben)

 © Laser & Co. Solutions GmbH

Strg + `-`	Selektiertes Objekt wird verkleinert (nur in bestimmten Ausgaben)
Strg + `*`	Ursprüngliche Objektgröße wird wiederhergestellt
Strg + Maustaste	Selektiertes Objekt wird innerhalb des Diagramms kopiert
Strg + Cursortasten:	
- Cursor nach links	Selektiertes Objekt wird in X-Richtung verkleinert
- Cursor nach rechts	Selektiertes Objekt wird in X-Richtung vergrößert
- Cursor nach oben	Selektiertes Objekt wird in Y-Richtung vergrößert
- Cursor nach unten	Selektiertes Objekt wird in Y-Richtung verkleinert
Enter	Editormodus zum Definieren der Objekte
Entf	Löschen
Cursortasten	Selektiertes Objekt wird verschoben (in Verbindung mit der Umschalttaste sind größere Schritte möglich)
`+`	Diagramm vergrößern
`-`	Diagramm verkleinern
`*`	Einpassen des Diagramms

Anhang: Tastenbelegung im Struktogramm

F3	öffnet den Quelltext
F5	startet das Programm (bei SiSy AVR das myAVR Controlcenter)
F8	Codegenerierung des aktuellen Struktogramms
F9	übersetzt das Struktogramm
ESC	Ebene nach oben
Num `+`	selektiertes Objekt wird vergrößert
Num `-`	selektiertes Objekt wird verkleinert
Enter	ermöglicht die Bearbeitung des Quelltextes im Struktogramm-Editor
Entf	löschen
T	ermöglicht die Bearbeitung des Titels
K	ermöglicht die Bearbeitung des Kommentars
B	Begin neue Funktion einfügen
S	Sequenz einfügen (Folge/Block)
D	Do einfügen (elementare Aktion)
W	While-Schleife einfügen (kopfgesteuerte Schleife)
R	Repeat-Schleife einfügen (fußgesteuerte Schleife)
F	For-Schleife einfügen (Zählschleife)
A	Alternative einfügen
C	Case einfügen (Fallunterscheidung)
I	If-Zweig einfügen
E	Exit einfügen
L	Loop einfügen
Cursortasten	Bewegung im Struktogramm

 © Laser & Co. Solutions GmbH

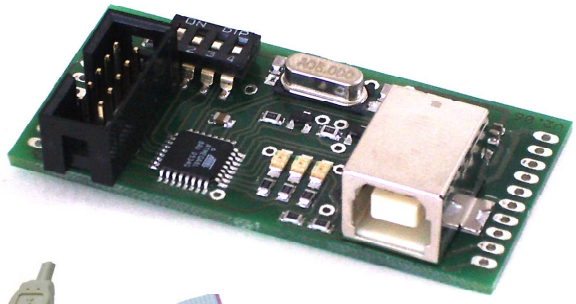
Anhang: Mausoperationen

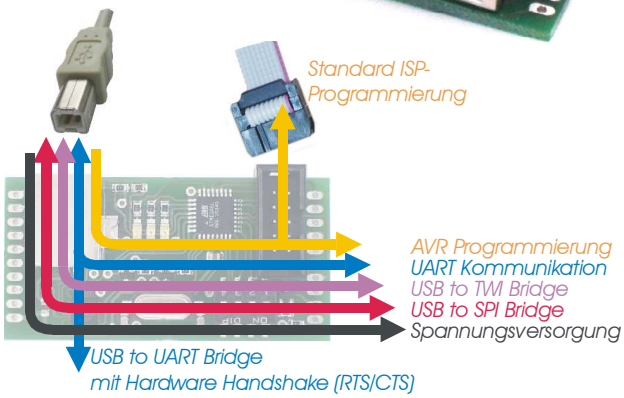
Die Maus hat in SiSy eine Anzahl von nützlichen Funktionen, welche die Arbeit in Projekten erleichtern.

Selektion	*Klick auf Objekt* Objekt ist markiert und kann separat weiterbearbeitet werden.
Selektion aufheben	*Klick auf Fensterhintergrund* Aufhebung der Objektmarkierung.
Mehrfachselektion	*Umschalttaste + Klick auf Objekt* Selektion/Markierung von mehreren Objekten zur Weiterbearbeitung. *Markise* Mit gedrückter linker Maustaste auf Fensterhintergrund und Ziehen eines Rechtecks über zu markierende Objekte.
Verschieben	*Drag & Drop im Diagramm* Objekt mit linker Maustaste anfassen und verschieben. Objekte werden am Raster verschoben. *Umschalttaste + Drag & Drop im Diagramm* Verschieben von Objekten ohne Raster.
Fensterinhalt schieben	*Linke und rechte Maustaste drücken + Verschieben der Maus im Diagramm* Der komplette Diagramminhalt wird geschoben.
Objekt kopieren	*STRG + Drag & Drop* Maustaste gedrückt halten und Mauszeiger vom Objekt auf den Fensterhintergrund führen. Eine Originalkopie des Objektes wird im aktuellen oder in einem anderen Diagramm erzeugt.
Referenz erzeugen	*Drag & Drop aus Navigator* Ziehen des gewünschten Objektes aus dem Navigator in das Diagramm. Es wird eine Referenz des gewählten Objektes erzeugt. *Drag & Drop aus Objektbibliothek* Rot beschriftete Objekte können nur als Referenz erzeugt werden. Eine Liste zur Auswahl des gewünschten Typs erscheint. *Strg + Drag & Drop aus Objektbibliothek* Eine Liste zur Auswahl der gewünschten Referenz des Originalobjektes erscheint.

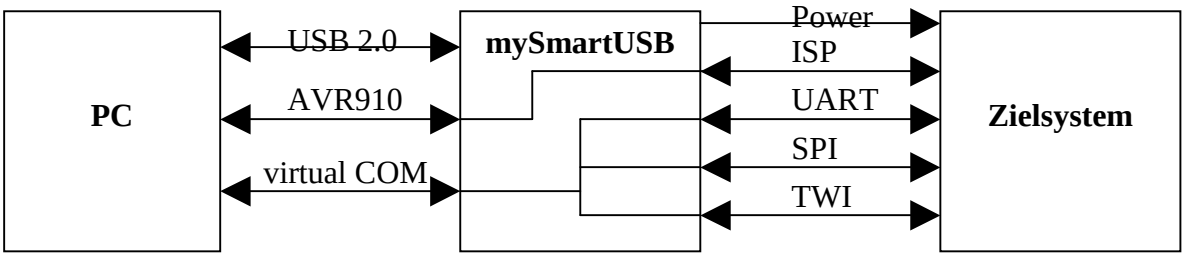
	Drag & Drop aus anderem Diagramm Ziehen des gewünschten Objektes aus dem Quelldiagramm in das Zieldiagramm. Es wird eine Referenz des gewählten Objektes erzeugt.
Objekt anlegen	*Drag & Drop aus Objektbibliothek* Ein Objekt aus der Objektbibliothek wird im Diagramm angelegt und steht zur Verfeinerung bereit.
Objekt anhängen	*Drag & Drop Verteiler auf Fensterhintergrund* Durch Ziehen einer Kante vom Verteiler auf den Fensterhintergrund wird ein neues Objekt erzeugt. Nach Auswahl des Objekttyps sind die Objekte miteinander verbunden.
Objekte verbinden	*Drag & Drop Verteiler zu Objekt* Klick auf den Verteiler des zu verbindenden Objektes. Bei gedrückter linker Maustaste auf das gewählte Objekt ziehen. *Verbindung aus Objektbibliothek (in der UML)* Hierbei wird erst die gewünschte Verbindung in der Objektbibliothek angeklickt und danach die beiden zu verbindenden Objekte im Diagramm nacheinander.
Verbindung anordnen	*Drag & Drop Mittelpunkt* Beliebige Gestaltung der Verbindung durch Ziehen mit der Maus.
Verbindung ändern	*Drag & Drop Anfangs-/Endpunkt einer Kante* Für die Verbindung wird ein neues Zielobjekt gewählt.
Objekt definieren	*Doppelklick auf Objekt* Durch Doppelklick auf Objekte öffnet sich das Kontextmenü. Bei Abschalten des Menüs unter *Einstellungen/Menü bei Doppelklick* erscheint eine Zeile zur Namensgebung. Mit ESC wird die Eingabe bestätigt. *Doppelklick auf Verteiler* Es wird der Definieren-Dialog aufgerufen, in dem das Objekt benannt und beschrieben werden kann.
Kontextmenü öffnen	*Klick mit rechter Maustaste auf Objekt*
Fenster neu zeichnen	*Doppelklick auf Fensterhintergrund* *Hinweis***:** Doppelklick mit linker Maustaste wirkt wie Enter.
Fenster aktualisieren	*Strg + Doppelklick auf Fensterhintergrund* Die vom Programm ausgeführten aber noch nicht sichtbargemachten Befehle werden im Fenster erstellt. Das Fenster wird aktualisiert.

- einfache Kommunikation mit dem PC oder Notebook über die USB-Schnittstelle
- über den ISP-Adapter können eine Vielzahl von AVR-Systemen programmiert werden,
- 10 PIN Atmel Standardbelegung
- Interface für das myAVR Board 2 USB
- Industriestandard-Controller (CP2102) USB zu seriell Konverter
- ATmega8 mit Firmware nach ATMEL AN910 und AN911 Programmierstandard
- Schnelle Programmierung über USB 2.0 durch Blockmode (AVR911)
- umschaltbar zwischen Programmier-, Kommunikations- und Ruhemodus
- Power On und Power Off durch Anwender schaltbar
- stellt virtuellen COM-Port zur Verfügung
- Zustandsanzeige (rote/grüne LEDs)
- Spannungsversorgung über USB-Anschluss
- einfache Handhabung
- updatefähig über Bootloader
- DIP-Schalter für Betriebsart, Power, Programmier- und Datenmodus, RESET und Bootloader





Blockbild / Übersicht



Allgemeine Beschreibung

mySmartUSB ist ein kompakter USB-Programmer und Kommunikationsinterface für Atmel AVR-Mikrocontroller. Sie können mit Hilfe dieses Boards eine Vielzahl von AVR-Systemen ganz einfach über die vorhandene USB-Schnittstelle programmieren oder mit den Systemen über eine UART-USB-Bridge kommunizieren. Es ist ein anschlussfertiges Modul in SMD-Bauweise. Er verfügt über spezielle Anschlussoptionen für das myAVR Board 2 USB und Standard-ISP. Die Kommunikation mit dem PC läuft völlig transparent über einen virtuellen COM-Port. Das Board wird per USB-Schnittstelle mit dem PC verbunden und mit Spannung versorgt. Es ist kein Parallel- oder COM-Port nötig, was gerade für Notebookbesitzer von Vorteil ist.

Hersteller: Laser & Co. Solutions GmbH
Promenadenring 8
02708 Löbau, Deutschland

© Laser & Co. Solutions GmbH

Technische Daten

Hardwareversion 2.06

Maximalwerte

Maximalspannung:	5,3 V über den USB-Bus
Maximalstrom:	100 mA über den USB-Bus
Lagertemperatur:	-20 ℃ bis +70 ℃

Betriebsdaten

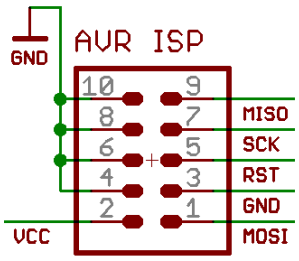
Versorgungsspannung:	5 V über den USB-Bus
Betriebsstrom:	10-20 mA typisch ohne weitere Verbraucher bis 100 mA bei Anschluss an Zielsysteme
Betriebsspannung:	5V
Betriebstemperatur:	0 ℃ bis +30 ℃

Schnittstellendaten

Pinbelegung ISP: Standard-ISP 10polig, Wannenstecker

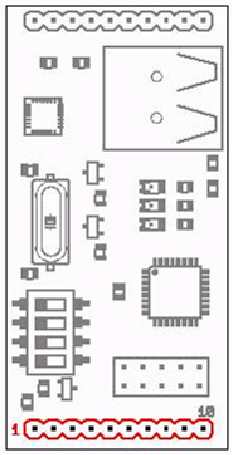
Pin 1 MOSI	Pin 6 GND
Pin 2 VCC	Pin 7 SCK
Pin 3 GND	Pin 8 GND
Pin 4 GND	Pin 9 MISO
Pin 5 RESET	Pin 10 GND

Programmierkabel: Flachbandkabel,
10polig 1:1,
Buchse-Buchse
10poliger Standard Atmel-ISP
Anschluss



mySmartUSB Interface: 10polig, Sockelleiste, gewinkelt, rechts, RM 2,54 mm,

Pin10	Masse
Pin 9	VCC für Board (schaltbar, maximale Belastung USB-Port beachten)
Pin 8	RxD PC (USB-UART-Bridge)
Pin 7	TxD PC (USB-UART-Bridge)
Pin 6	Reset für Board
Pin 5	SPI-MOSI
Pin 4	SPI-MISO
Pin 3	SPI-SCK
Pin 2	SPI-SS
Pin 1	Reset für mySmartUSB (nicht benutzen, offen lassen)

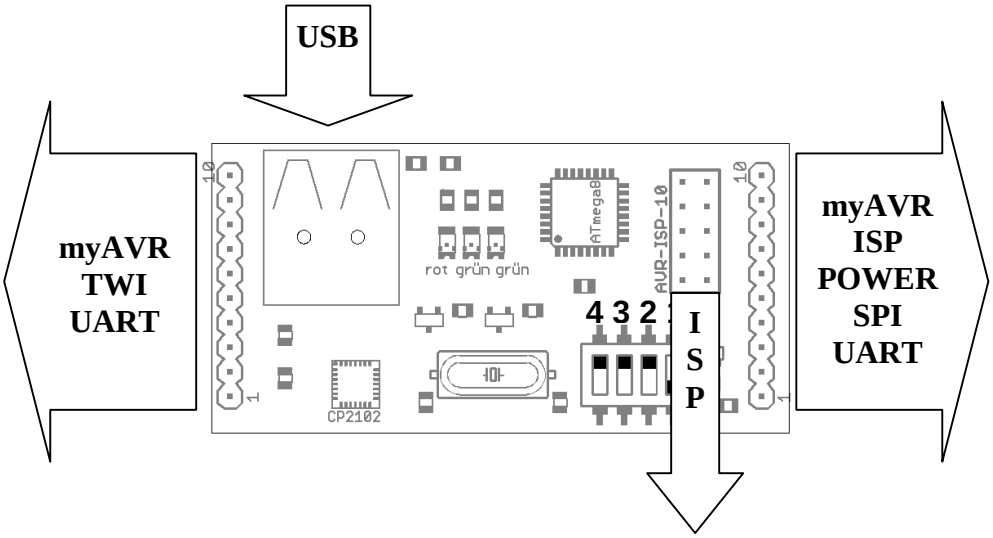


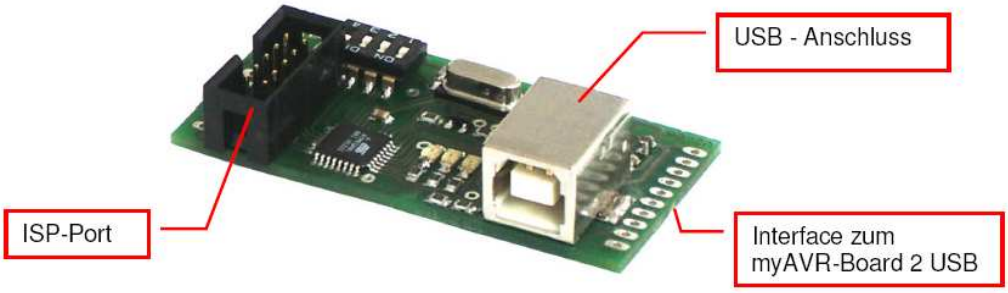
Mechanische Daten

Abmaße (L x B x H):	ca. 60 mm x 30 mm x 12 mm
Gewicht:	ca. 12 g
Rastermaß:	2,54 mm

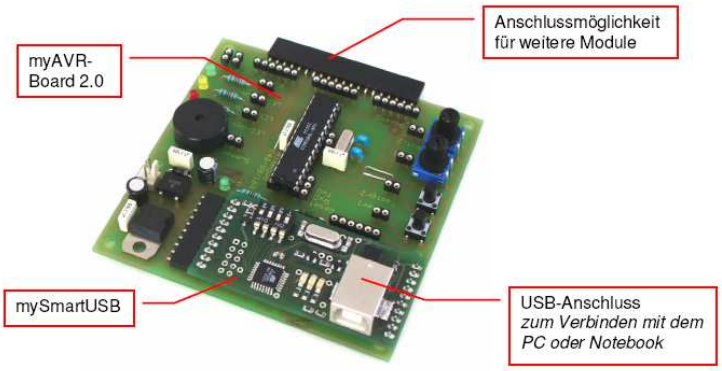
Leiterplattenmaterial: FR8, 1,5 mm Dicke, 0,35 µm Cu Auflage, zweiseitig, Lötstoppmaske, verzinnt, durchkontaktiert

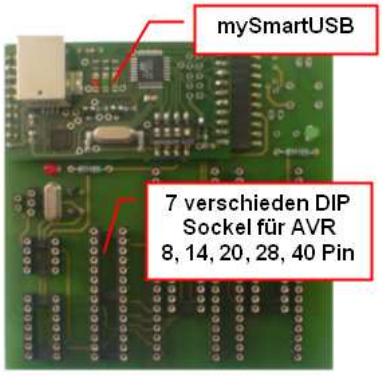
Funktionsschema

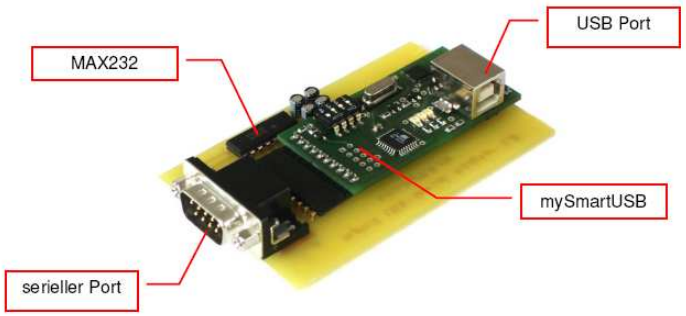




Anwendungsbeispiele auf dem myAVR Board, als COM-USB-Adapter und auf dem myMultiProg USB







Das mySmartUSB Protokoll
Die Firmware des mySmartUSB verfügt neben den AVR910 und AVR911 spezifischen Protokollen über ein eigenes Protokoll um die zusätzlichen Funktionen des mySmartUSB anzusprechen. Das mySmartUSB Protokoll wird über eine Kennung aktiviert die den eigentlichen Kommandos als Präfix vorangestellt werden muss.

mySmartUSB-Kommandos:

Baudrate:	19200,8,n,1	
Präfix/Format:	"æµº123©"+Kommando als ASCII-Zeichen (= 0xE6 0xB5 0xBA 0xB9 0xB2 0xB3 0xA9)	
Kommandos:	r	Reset Board
	R	Reset Programmer
	+	Board-Power-On
	-	Board-Power-Off
	p	Programmier-Modus
	d	Daten-Bypass/Durchreich-Modus RS232
	q	Quite-Modus
	i	Status wird zurückgegeben

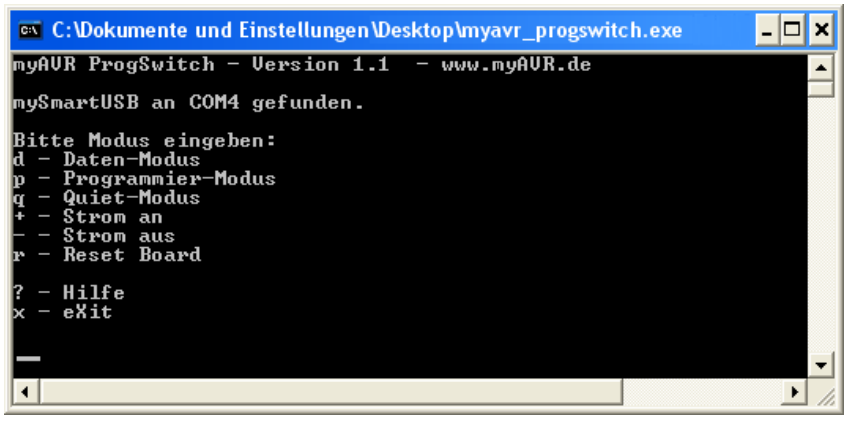
Das Kommandozeilen-Tool *myavr_progswitch*
Im Downloadbereich von www.myAVR.de steht Ihnen das Programm *myavr_progswitch.exe* zur Verfügung. Mit diesem Programm kann zwischen den einzelnen Arbeitsmodi des mySmartUSB-Programmers gewechselt werden. Das Umschalten der Modi kann per Aufruf-Parameter und über einen Dialog gesteuert werden. Das Programm ermöglicht des Weiteren das Brennen von HEX-Dateien und kann über Make-Dateien aufgerufen werden.

Hinweis: Verwenden Sie die aktuelle Firmware des mySmartUSB

Kommandos.
Parameter:

-m<MODE>	Angabe des Arbeitsmodus (p|d|q = Programming|Data|Quiet).
-p<on|off>	Angabe ob Stromversorgung (de)aktiviert werden soll.
-r	Reset des Boards
-b "Dateiname"	Brennen einer HEX-Datei

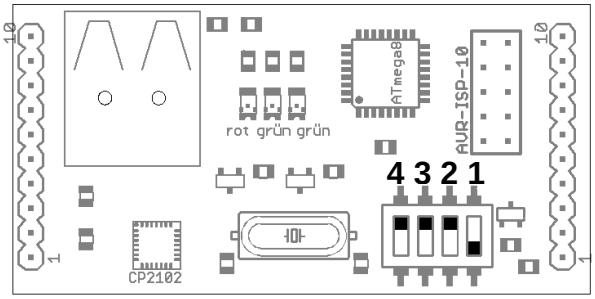
Ohne Parameter startet das Programm im Dialog-Modus. Hier kann beliebig zwischen den Modi hin- und hergeschaltet werden.



DIP-Schalter und Betriebsmodi

Standardeinstellung der DIP-Schalter

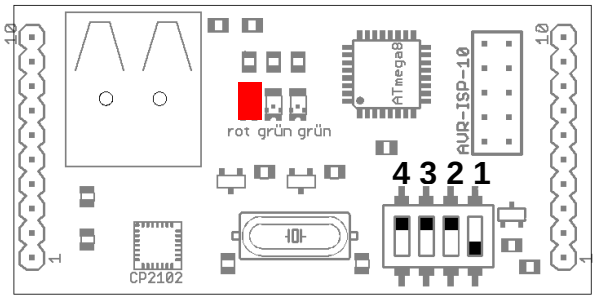
- Soft-Mode (per Software einstellbar)
  - o Programmiermodus (AVR910)
  - o Datenmodus (UART-USB-Bridge)
  - o stumm



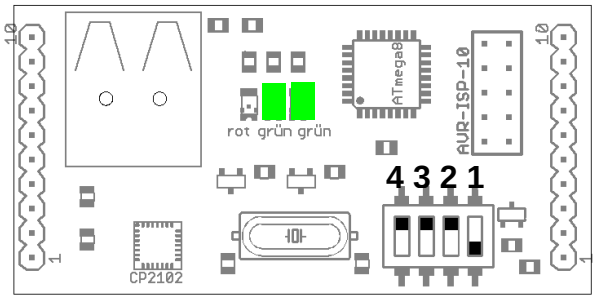
Hinweis: Die beschrieben Modi funktionieren nur im Zusammenhang mit dem myAVR Board 2. Für Fremdsysteme kann mySmartUSB als AVR 910 kompatibler Programmer verwendet werden (AVR-ISP Anschluss, 10polig).

LED Statusanzeige:

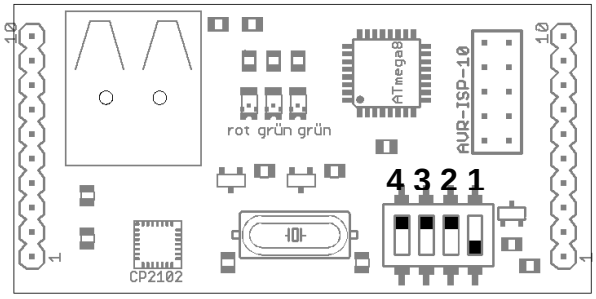
- Programmiermodus (AVR910)
  - o rote LED ON
  - o grüne LEDs flackern bei Datenübertragung (Programmierung)



- Datenmodus (UART-USB-Bridge)
  - o grüne LEDs ON
  - o rote LED flackert bei Datenübertragung

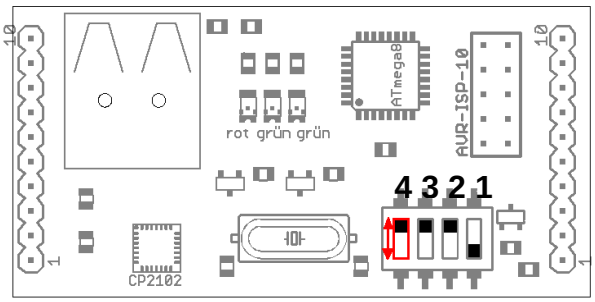


- stumm
  - o alle LEDs OFF



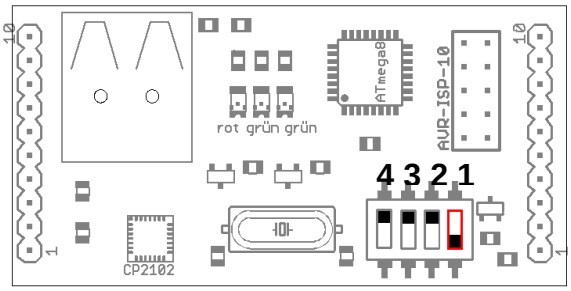
RESET des Adapters

- DIP 4 kurz ON/OFF schalten
- danach wieder Standardeinstellung

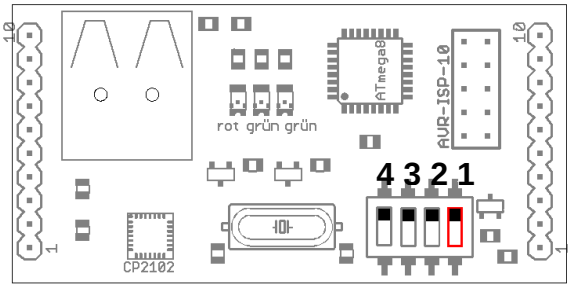


Enable Power

- DIP 1 ON (Standard)
- erlaubt das Ein- und Ausschalten der Boardspannung per Firmware
- Zielsystem wird über USB gespeist **Achtung: I < 80 mA**

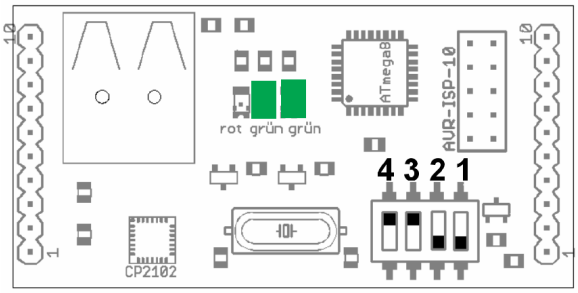


- DIP 1 OFF
- unterdrückt das Ein- und Ausschalten der Boardspannung per Firmware
- Zielsystem benötigt eine eigene Stromversorgung



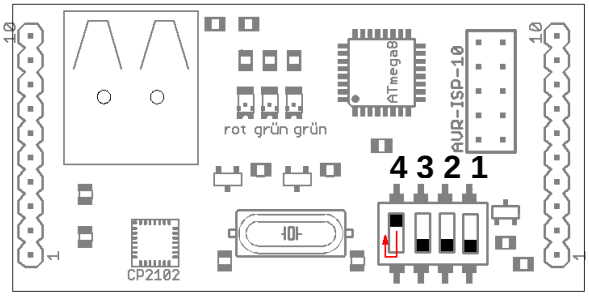
Datenmodus erzwingen
(UART-USB-Brigde)

- DIP 2 ON
- DIP 3 OFF
- DIP 4 OFF



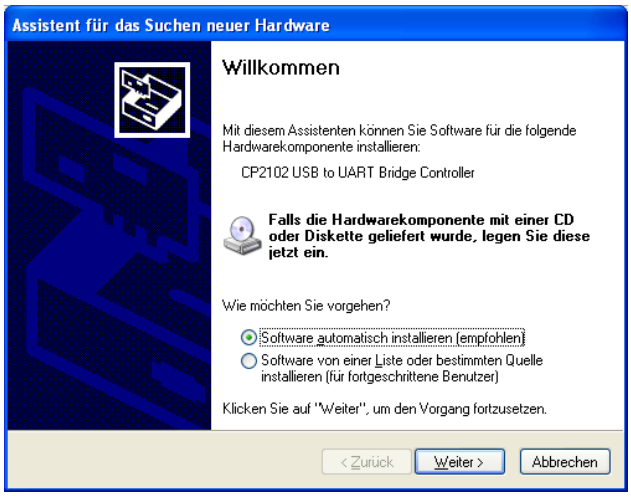
Firmware Update

- DIP 2 ON
- DIP 3 ON
- DIP 4 kurz ON/OFF = RESET
- startet den Bootloader und erwartet die myAVR Updatekennung

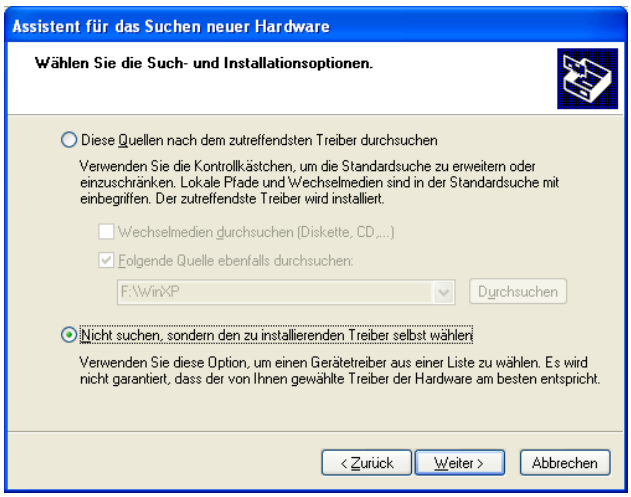


Treiberinstallation

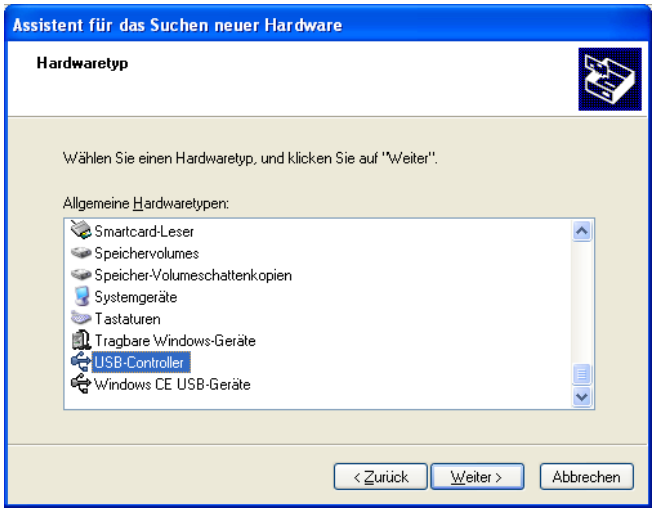
Laden Sie den Treiber für mySmartUSB (CP2102) von www.myAVR.de Bereich „Downloads“. Entpacken Sie den Treiber in einem temporären Verzeichnis oder auf dem Desktop. USB Adapter an den PC/Notebook anschließen. Danach erscheint folgender Dialog und wählen Sie: „Software von einer Liste oder bestimmten Quelle installieren“


Assistent für das Suchen neuer Hardware
Willkommen
Mit diesem Assistenten können Sie Software für die folgende Hardwarekomponente installieren:
CP2102 USB to UART Bridge Controller
Falls die Hardwarekomponente mit einer CD oder Diskette geliefert wurde, legen Sie diese jetzt ein.
Wie möchten Sie vorgehen?
Software automatisch installieren (empfohlen)
Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)
Klicken Sie auf "Weiter", um den Vorgang fortzusetzen.
< Zurück
Weiter >
Abbrechen

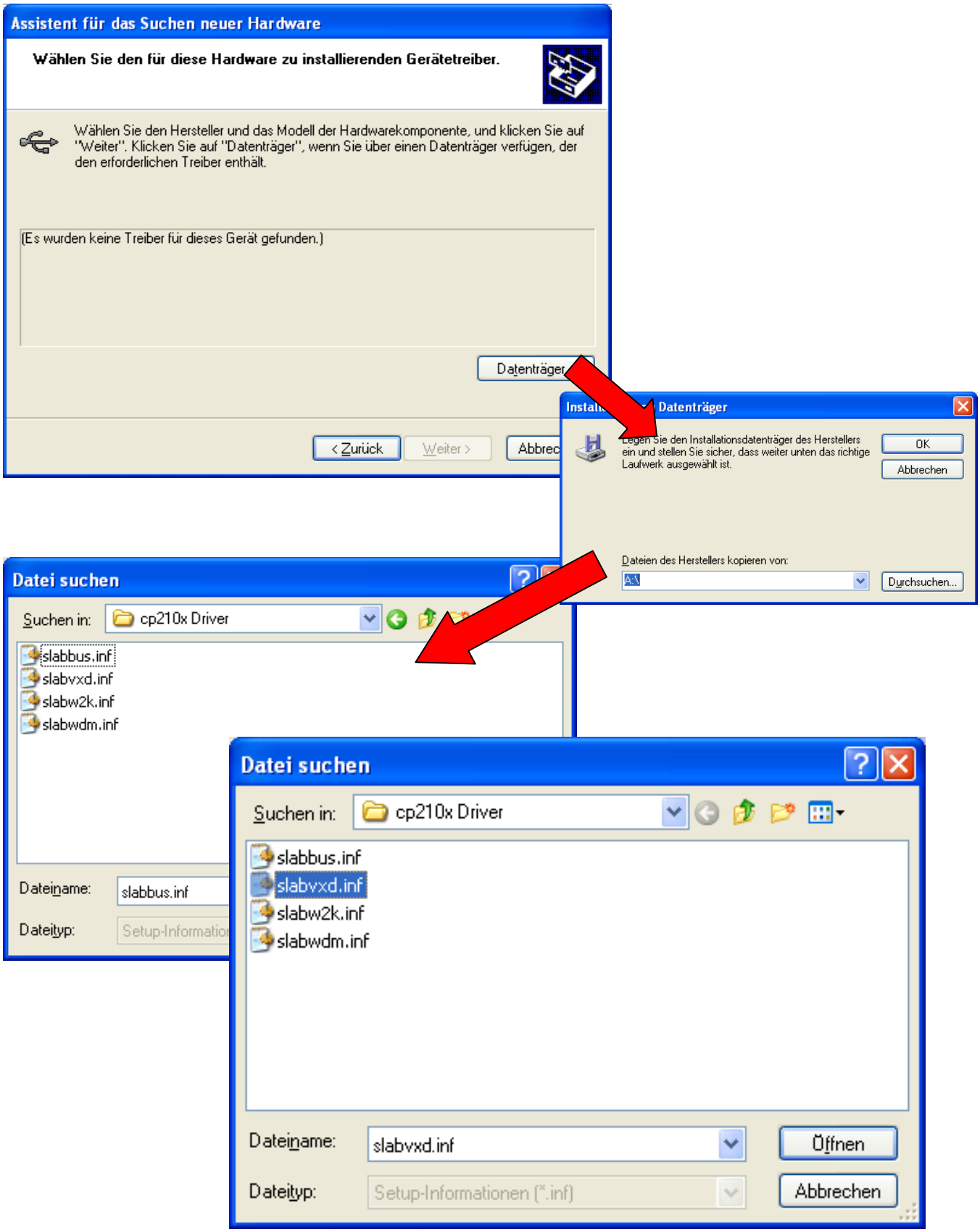
Im folgenden Dialog wählen Sie: „Nicht suchen“

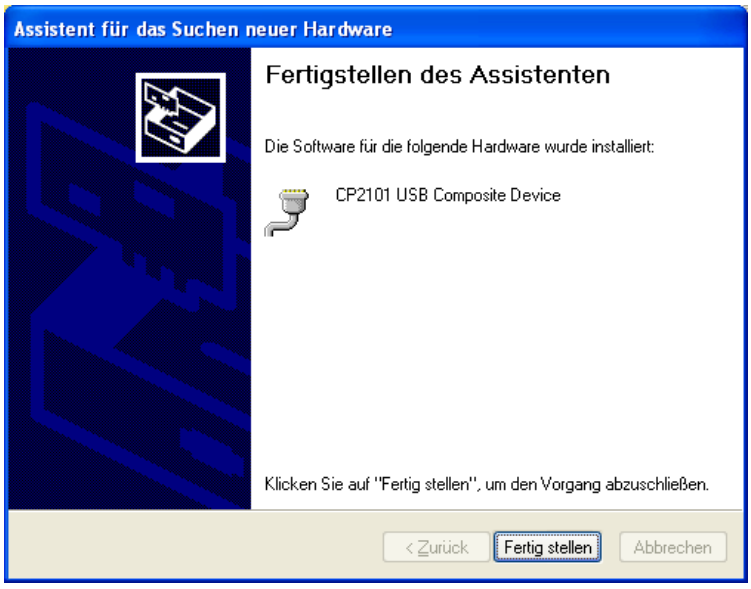

Assistent für das Suchen neuer Hardware
Wählen Sie die Such- und Installationsoptionen.
Diese Quellen nach dem zutreffendsten Treiber durchsuchen
Verwenden Sie die Kontrollkästchen, um die Standardsuche zu erweitern oder einzuschränken. Lokale Pfade und Wechselmedien sind in der Standardsuche mit einbegriffen. Der zutreffendste Treiber wird installiert.
Wechselmedien durchsuchen (Diskette, CD,...)
Folgende Quelle ebenfalls durchsuchen:
F:\WinXP
Durchsuchen
Nicht suchen, sondern den zu installierenden Treiber selbst wählen
Verwenden Sie diese Option, um einen Gerätetreiber aus einer Liste zu wählen. Es wird nicht garantiert, dass der von Ihnen gewählte Treiber der Hardware am besten entspricht.
< Zurück
Weiter >
Abbrechen

Als Hardwaretyp wählen Sie „USB-Controller“

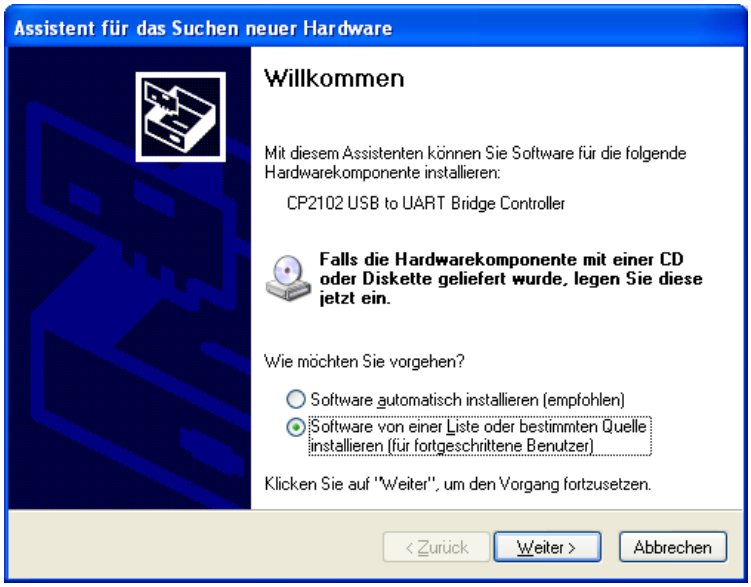


Der Treiber wird nicht automatisch gefunden sondern muss über die Schaltfläche „Datenträger“ gesucht werden

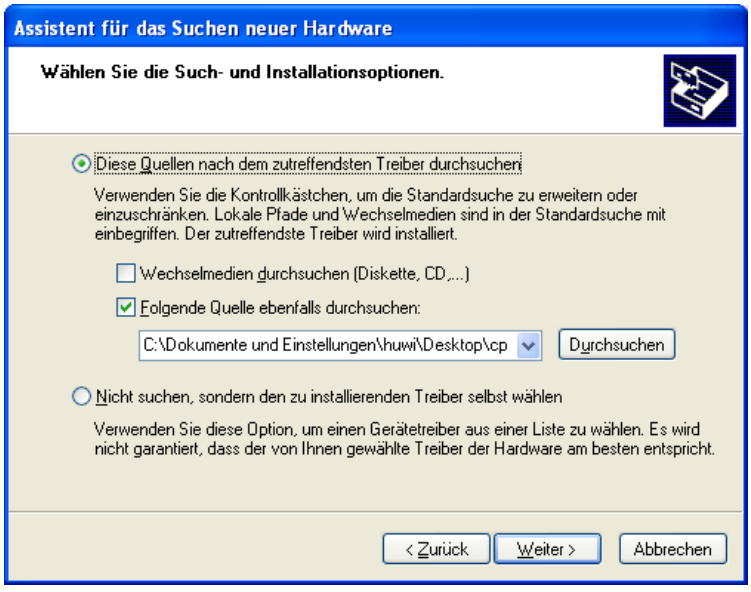


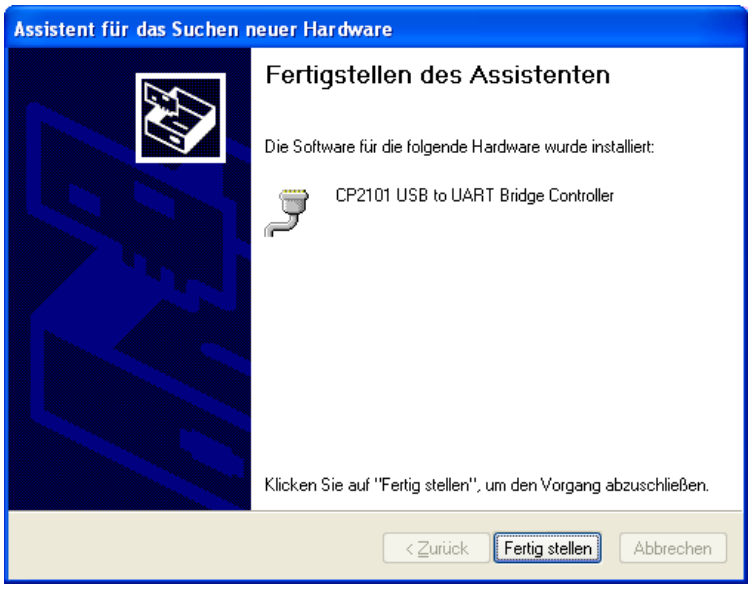

Assistent für das Suchen neuer Hardware
Fertigstellen des Assistenten
Die Software für die folgende Hardware wurde installiert:
CP2101 USB Composite Device
Klicken Sie auf "Fertig stellen", um den Vorgang abzuschließen.
< Zurück
Fertig stellen
Abbrechen

Danach muss ein zweiter aber verkürzter Durchlauf der Treiberinstallation erfolgen um den virtuellen COM-Port zu installieren (USB-UART-Bridge).

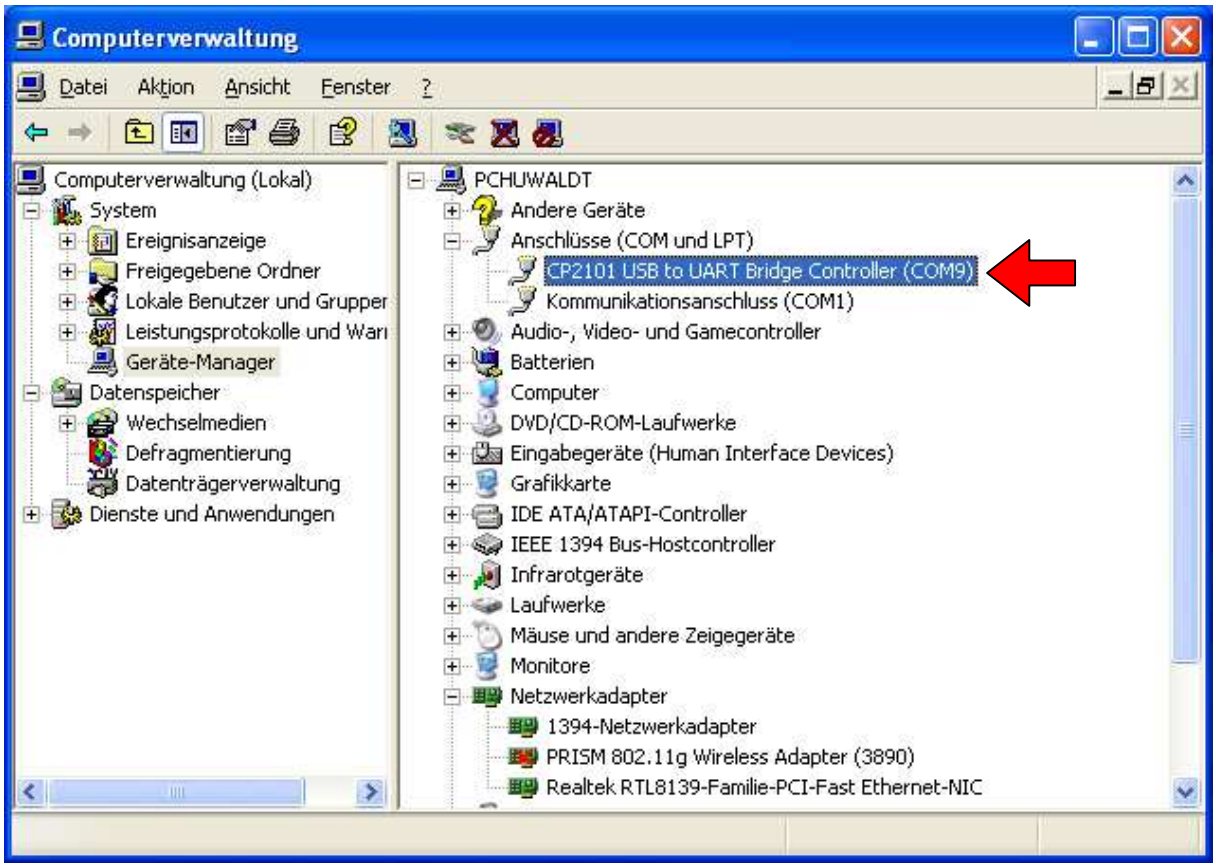

Assistent für das Suchen neuer Hardware
Willkommen
Mit diesem Assistenten können Sie Software für die folgende Hardwarekomponente installieren:
CP2102 USB to UART Bridge Controller
Falls die Hardwarekomponente mit einer CD oder Diskette geliefert wurde, legen Sie diese jetzt ein.
Wie möchten Sie vorgehen?
Software automatisch installieren (empfohlen)
Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)
Klicken Sie auf "Weiter", um den Vorgang fortzusetzen.
< Zurück
Weiter >
Abbrechen

Die Treiberquelle ist bereits voreingestellt!


Assistent für das Suchen neuer Hardware
Wählen Sie die Such- und Installationsoptionen.
Diese Quellen nach dem zutreffendsten Treiber durchsuchen
Verwenden Sie die Kontrollkästchen, um die Standardsuche zu erweitern oder einzuschränken. Lokale Pfade und Wechselmedien sind in der Standardsuche mit einbegriffen. Der zutreffendste Treiber wird installiert.
Wechselmedien durchsuchen (Diskette, CD,...)
Folgende Quelle ebenfalls durchsuchen:
C:\Dokumente und Einstellungen\huwi\Desktop\cp
Durchsuchen
Nicht suchen, sondern den zu installierenden Treiber selbst wählen
Verwenden Sie diese Option, um einen Gerätetreiber aus einer Liste zu wählen. Es wird nicht garantiert, dass der von Ihnen gewählte Treiber der Hardware am besten entspricht.
< Zurück
Weiter >
Abbrechen

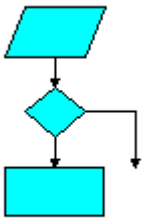

Assistent für das Suchen neuer Hardware
Fertigstellen des Assistenten
Die Software für die folgende Hardware wurde installiert:
CP2101 USB to UART Bridge Controller
Klicken Sie auf "Fertig stellen", um den Vorgang abzuschließen.
< Zurück
Fertig stellen
Abbrechen

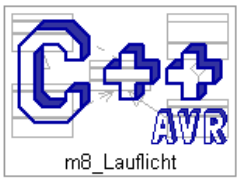
Jetzt ist die Installation abgeschlossen. Kontrollieren Sie im Gerätemanager unter welchem COM-Port sich der Controller angemeldet hat.



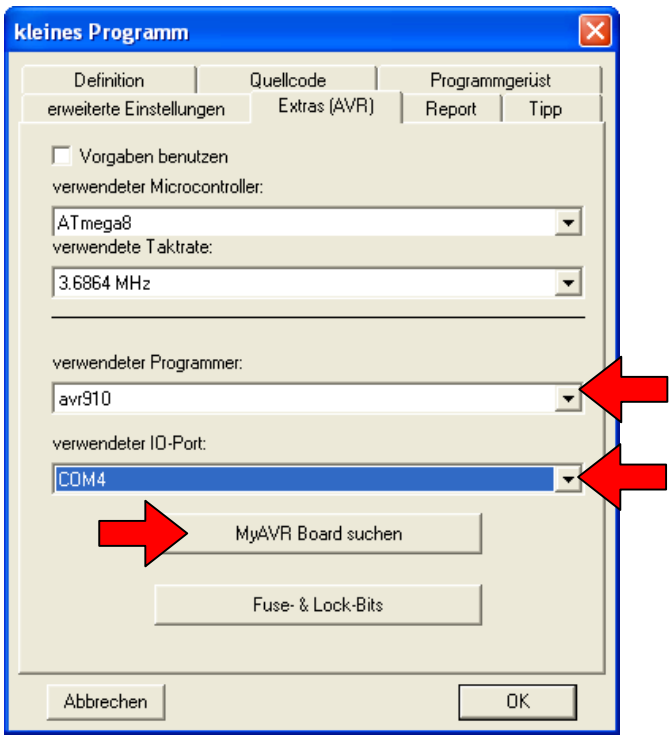
Anwendung mit SiSy (ab Version 2.17e)
Bei der Anwendung mit SiSy sind auf jedem Programmobjekt (kleines Programm, Programmablaufplan, Klassendiagramm) über den Dialog „Definieren“ folgende Einstellungen vorzunehmen: AVR910 oder AVR911, COMx (siehe Gerätemanager)

m8_Lauflicht


m8_Lauflicht

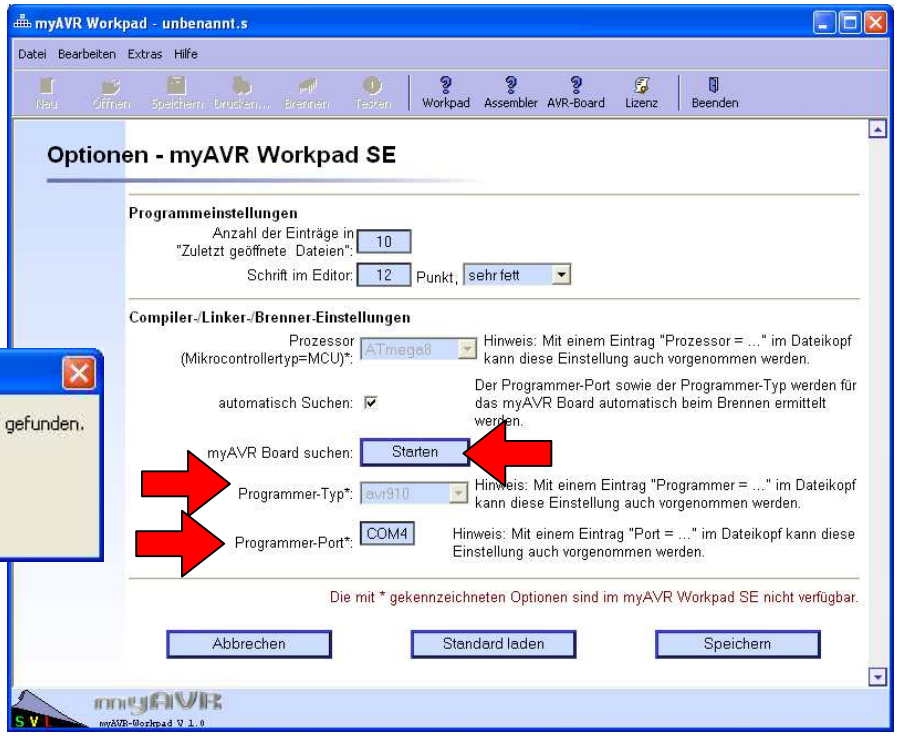


m8_Lauflicht



Sie können den Programmer auch automatisch suchen lassen.

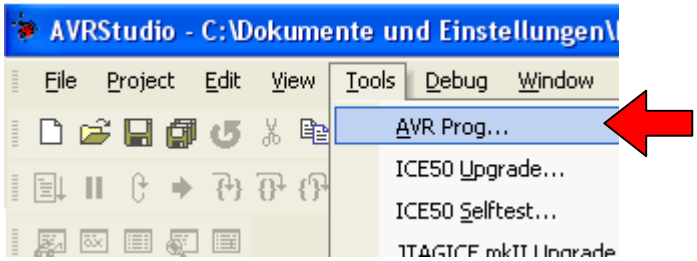
Anwendung mit dem myAVR Workpad
Im myAVR-Workpad finden Sie die Einstellungen des Programmers unter dem Menüpunkt „Extras / Einstellungen“. Es ist der Programmertyp AVR910 und der entsprechende COM-Port einzustellen.

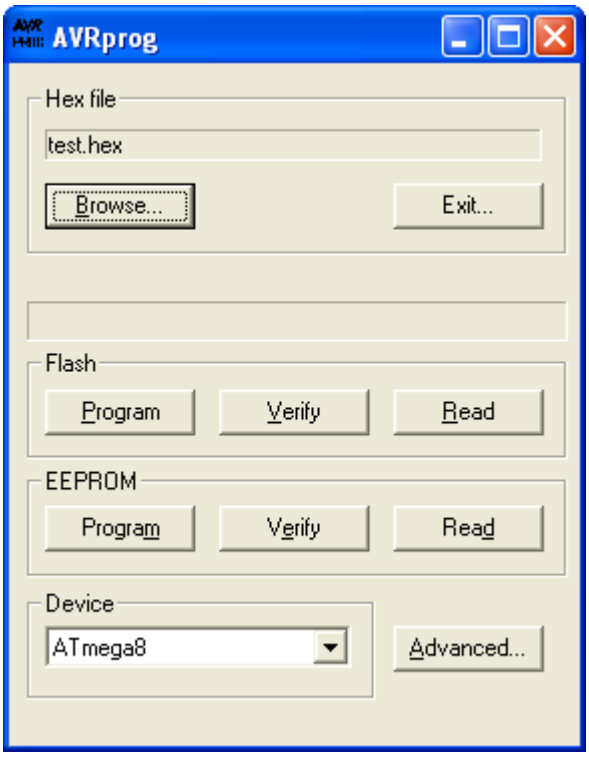


Sie können den Programmer auch automatisch suchen lassen.

Anwendung mit dem AVR Studio

Bei Verwendung mit dem AVR-Studio benutzen Sie „AVR Prog“ unter dem Menüpunkt Tools. Der Programmer muss bei Aufruf des Menüpunktes angesteckt sein. AVR Prog findet mySmartUSB automatisch.





AVR-Prog sucht standardmäßig nur an den COM-Ports 1 bis 4. Sollte der virtuelle COM-Port des mySmartUSB höher liegen findet das AVR Studio den Programmer nicht. In dem Fall muss über die Systemsteuerung ein für das AVR-Studio gültiger COM-Port zugewiesen werden. Dazu benötigen Sie Administratorrechte.

Gehen Sie wie folgt vor:

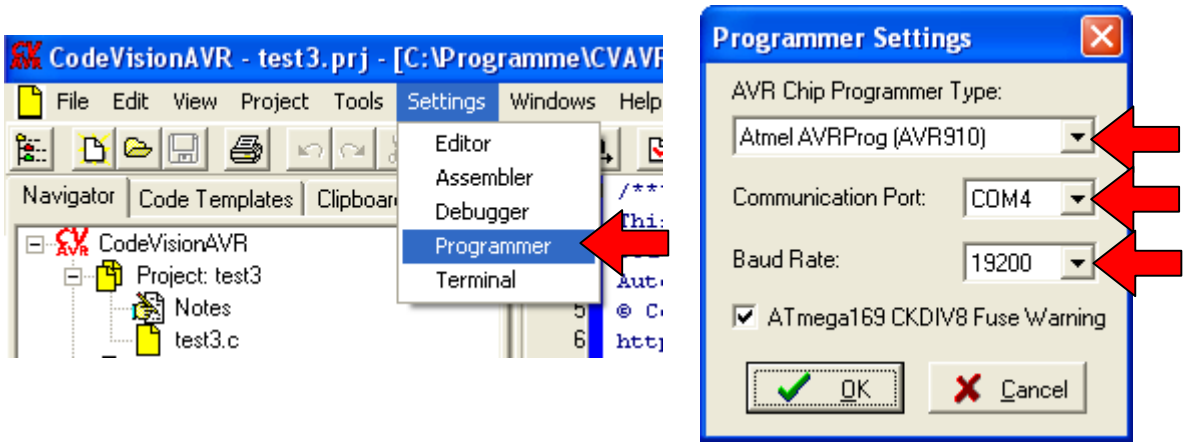
- Gerätemanager öffnen
- Rubrik: Anschlüsse (COM und LPT) bzw. USB-Controller öffnen
- Eintrag: "CP210x USB to UART Bridge Controller (COMx)" auswählen
- rechte Maustaste -> "Eigenschaften" -> "Anschlusseinstellungen" -> "Erweitert"
- "COM-Anschlussnummer" ändern und bestätigen
- Empfohlen ist COM3 oder COM4, beachten Sie mögliche Gerätekonflikte

Anwendung mit AVRDUDE
AVRDUDE ist eine Konsolenanwendung. Die Parameterzeile für AVRDUDE sollte wie folgt aussehen:

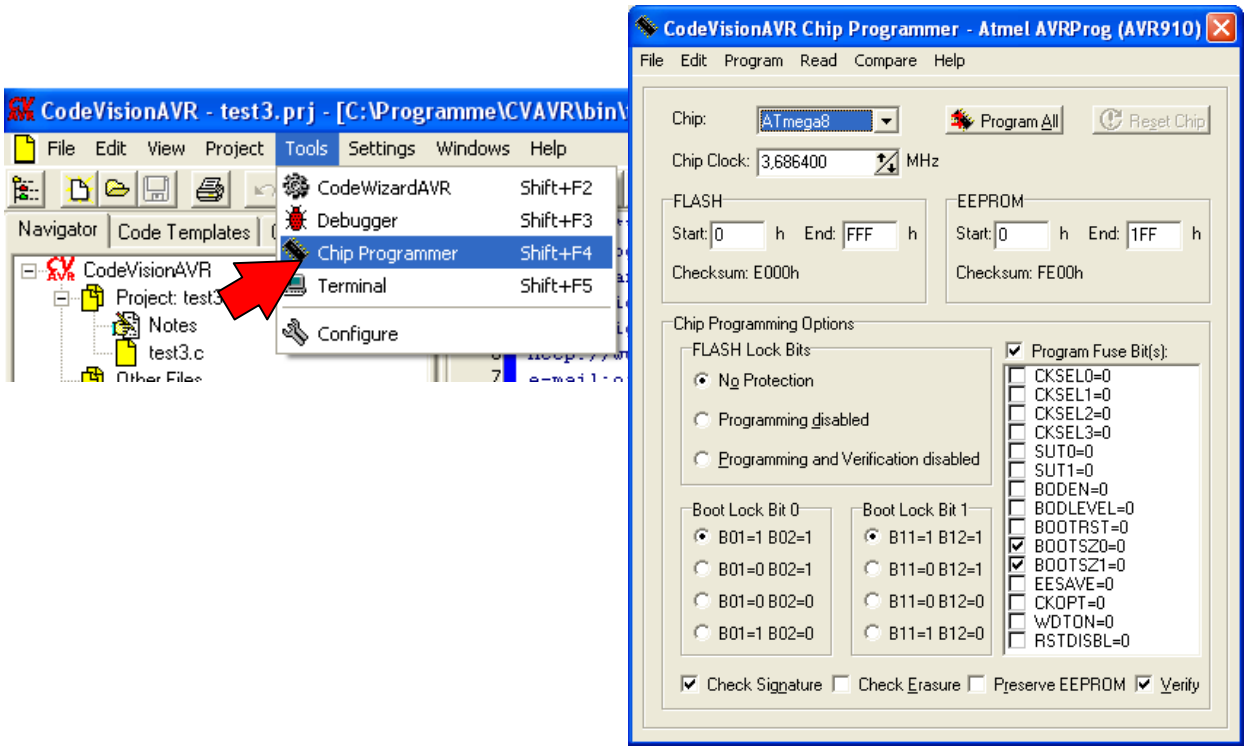
```
C:\Programme\winavr\bin\avrdude.exe -p ATmega8 -e -c avr910
     -P com4 -Uflash:w:"TEST.hex" :i

-p MCU-Typ
-e
-c avr910
-P COMx
-U flash:w:“Dateiname.hex“:i
```

Anwendung mit CodeVison
Die Programmereinstellungen erfolgen in CodeVision unter dem Menüpunkt „Settings / Programmer“. Die Einstellungen sind: Atmel AVR Prog (AVR910), COMx, 19200 Baud

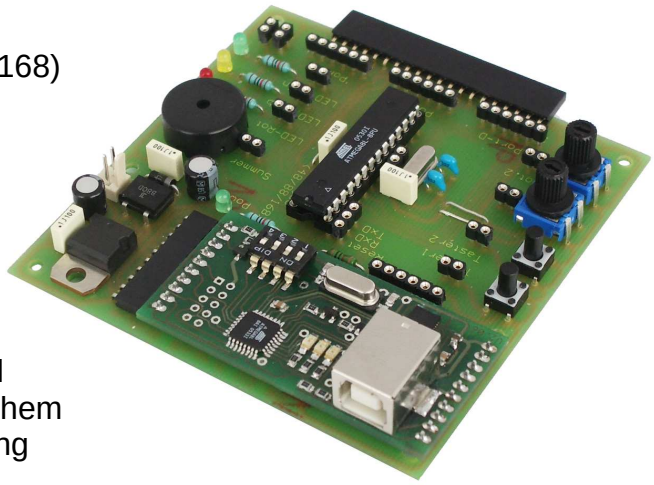


Der Programmer selbst wird über den Menüpunkt „Tools / Chip Programmer“ aufgerufen.



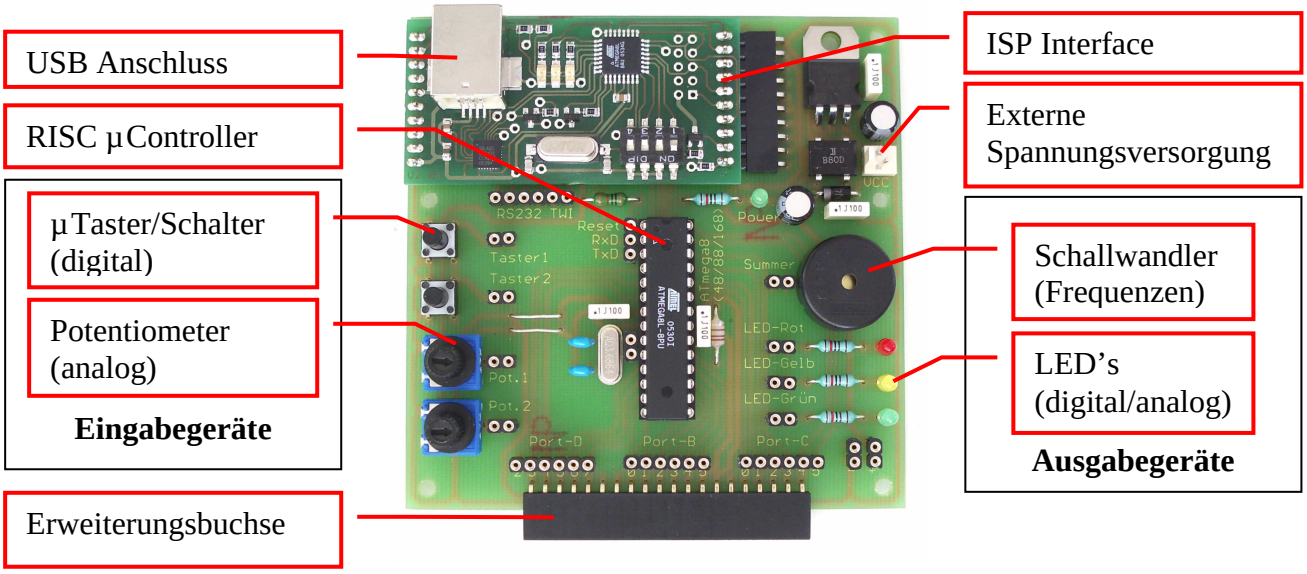
Sicherheitshinweise

Bei vorschriftsmäßigem Anschluss und Betrieb treten keine lebensgefährlichen Spannungen auf. Beachten Sie trotzdem die Vorschriften, die beim Betrieb elektrischer Geräte und Anlagen Gültigkeit haben. Wir versichern, dass die Leiterplatte durch den Hersteller getestet wurde. Für fehlerhaften und/oder vorschriftswidrigen Einsatz des Board übernehmen wir keine Garantie. Zum Anschluss des Boards an den PC ist eine Standard-USB-Kabel A-B zu verwenden. Der Einsatz anderer Kabel führt zu Fehlern. Für die ISP-Programmierung ist nur ein dafür vorgesehenes Kabel zu verwenden. Es dürfen keine Zielsysteme angeschlossen werden die mehr als 80 mA Stromaufnahme aufweisen.



- Lern- und Experimentierboard für *ATMEL* Mikrocontroller (ATmega8/48/88/168)
- Mit Controller und typischen Ein- und Ausgabegeräten (Taster, LEDs, usw.)
- Material: FR4, 1,5 mm; 0.35 µm Cu
- Leiterplatte gebohrt, verzinnt, Industriefertigung, robust, bedruckt
- Buchsenleiste für den Anschluss von weiteren Add-Ons
- Integrierter USB Programmer, kompatibel zum Atmel AN910 Protokoll und zusätzlichem RS232 Interface über die selbe Verbindung
- Duale Spannungsversorgung über USB oder externe Spannungsversorgung
- Besonders als Bausatz geeignet, da alle SMD Teile bereits bestückt sind
- Einfache Handhabung, keine Spezialkabel nötig
- Programmierbar in Assembler, C, C++

Blockbild / Übersicht



Allgemeine Beschreibung

Das myAVR Board 2.0 verfügt über einen RISC AVR-Mikrocontroller (ATmega8) der Firma ATMEL. Auf dem Board ist ein USB Programmer und Kommunikations-Port integriert. Des Weiteren befinden sich bereits einige typische Ein- und Ausgabegeräte wie zum Beispiel Potentiometer, Schalter, Frequenzwandler und LEDs auf dem Board. Die für das Board vorgesehenen Controller gehören zur Reihe der Mega-AVRs (ATmega8/48/88/168) und verfügen über alle wesentlichen Baugruppen. Das System ist nach didaktischen Gesichtspunkten für Ausbildung und Selbststudium konzipiert.

©Laser & Co. Solutions GmbH

Technische Daten

Betriebsdaten

Versorgungsspannung:	empfohlen 9V stabilisierte Gleichspannung
Betriebsstrom:	10-50 mA typisch ohne weitere Verbraucher bis 150 mA bei Verwendung des LCD Add-Ons
Betriebsspannung:	3,3 – 5,3 V
Betriebstemperatur:	0 ℃ bis +30 ℃

Maximalwerte

Versorgungsspannung:	12V
Maximalstrom:	500 mA Spitze, max. 300 mA kontinuierlich
Maximalspannung:	5,5 V
Lagertemperatur:	-20 ℃ bis +70 ℃

Schnittstellendaten

Programmierkabel und Kommunikationskabel:
Standard-USB-Kabel A-B

Erweiterungsport: Buchsenleiste, 20polig, 1reihig, Rastermaß 2,54 mm

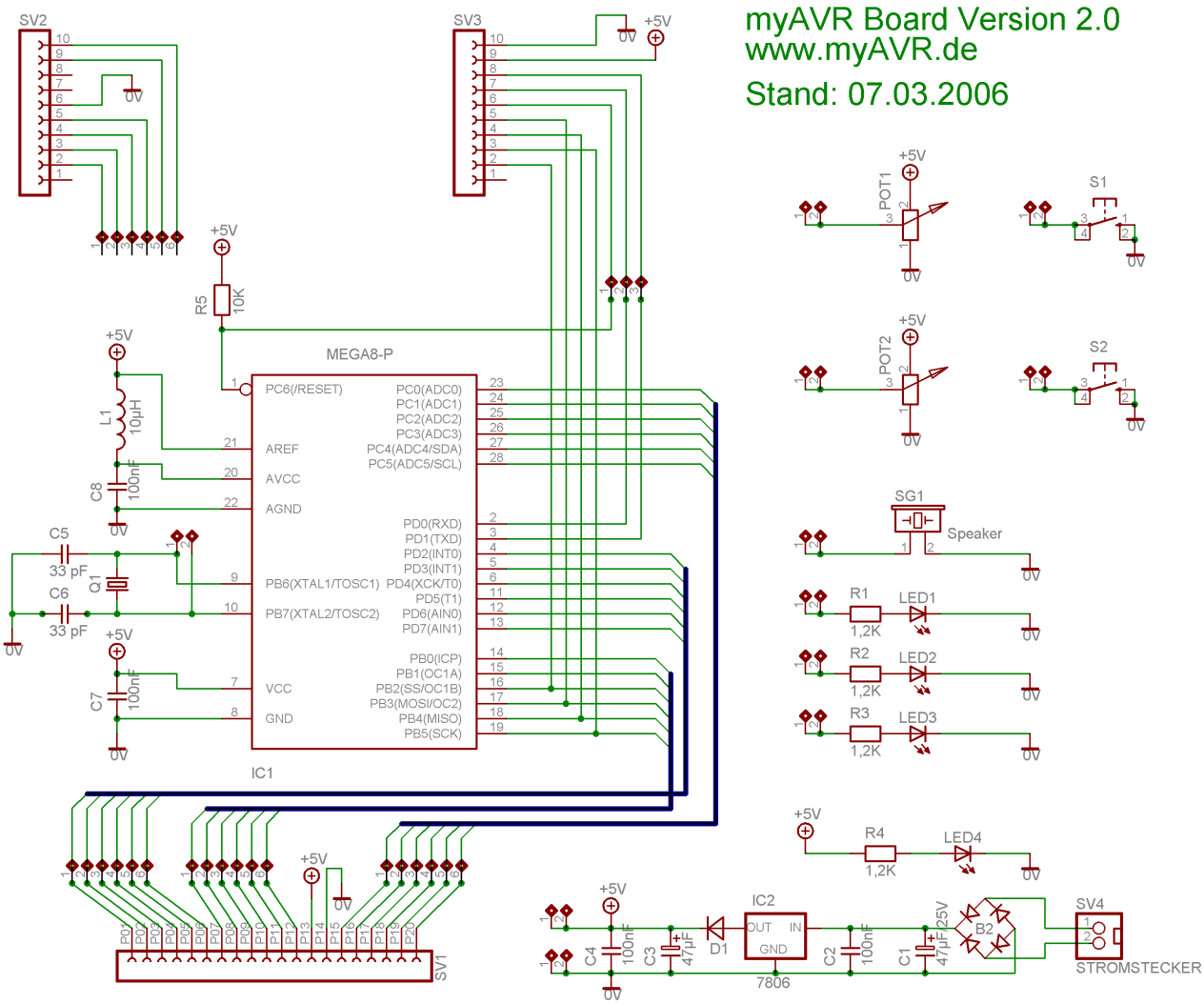
Pinbelegung:

1 = Port D.2	7 = Port B.0	15 = Port C.0
2 = Port D.3	8 = Port B.1	16 = Port C.1
3 = Port D.4	9 = Port B.2	17 = Port C.2
4 = Port D.5	10 = Port B.3	18 = Port C.3
5 = Port D.6	11 = Port B.4	19 = Port C.4
6 = Port D.7	12 = Port B.5	20 = Port C.5
13 = Boardspannung		14 = Masse

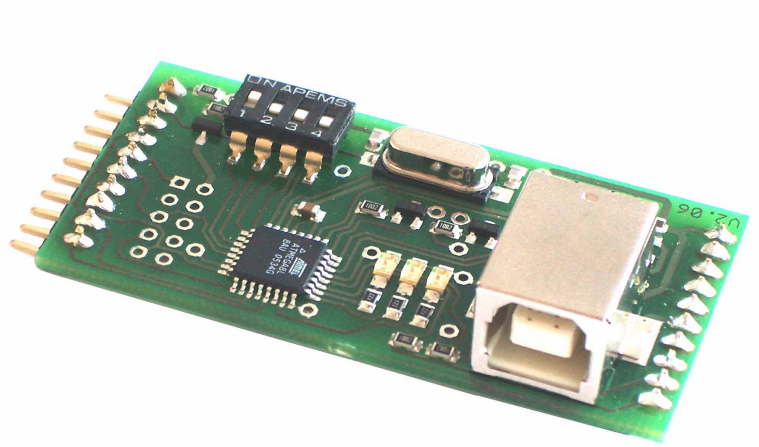
Mechanische Daten

Abmaße (L x B x H):	ca. 90 mm x 90 mm x 18mm
Gewicht:	ca. 50 g
Rastermaß:	2,54 mm
Leiterplattenmaterial:	FR8, 1,5 mm Dicke, 0,35 µm Cu Auflage, einseitig, Lötstoppmaske, verzinnt, Dokumentationsdruck, Abmessung 90 mm x 90 mm, bleifrei
Tochterplatine:	FR8, 1,5 mm Dicke, 0,35 µm Cu Auflage, zweiseitig, bleifrei, SMD bestückt,
Abmaße (L x B x H):	ca. 60 mm x 30 mm x 15 mm
Gewicht:	ca. 30 g
Rastermaß:	2,54 mm, für Komponenten in Printmontage
USB-Controller:	CP2102, SiliconLabs

Schaltplan



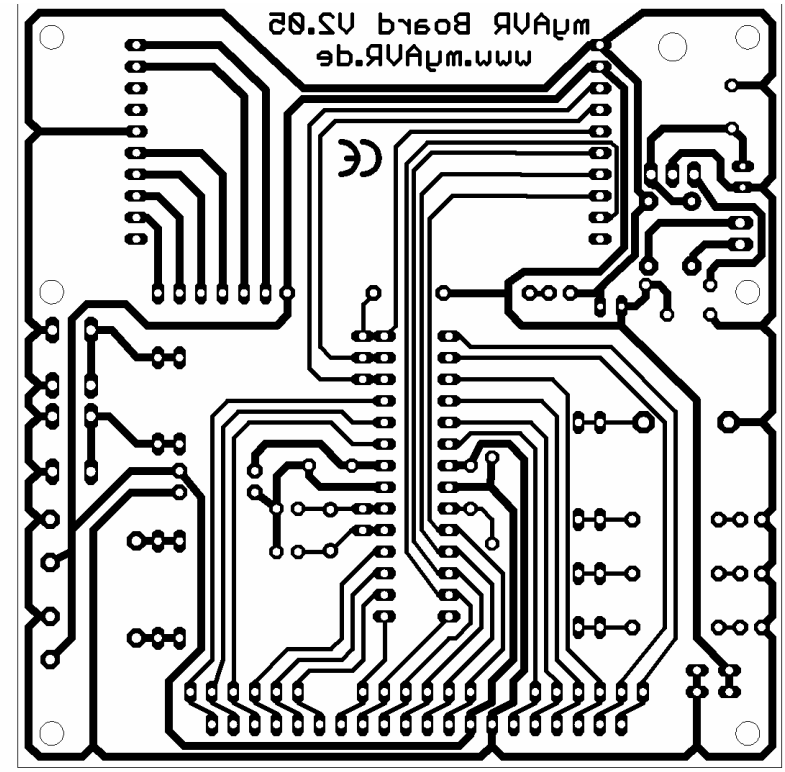
USB-Programmer und Interface (Tochterplatine)



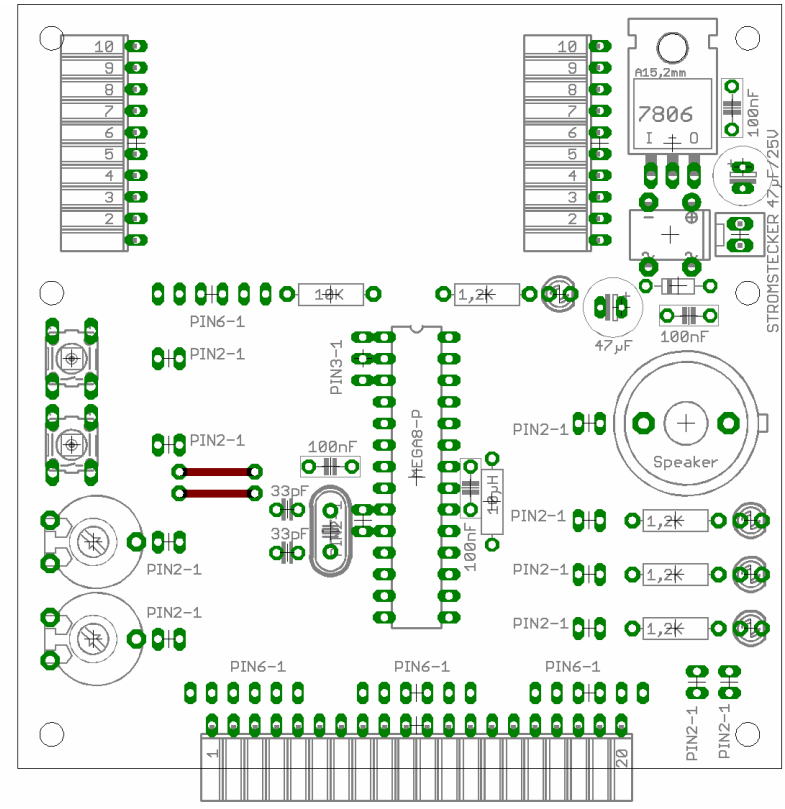
Der USB Programmer ist in SMD-Bauweise ausgeführt. Alle SMD Bauelemente sind bestückt. Der Programmer wird als Tochterplatine über zwei Buchsenleisten auf dem myAVR-Board integriert.

©Laser & Co. Solutions GmbH

Layout



Bestückungsplan



Stückliste

Material	Benennung / Bestellnummer	Stückzahl
Prozessor	ATmega8L 8PI	1
Spannungsregler	µA 7806	1
Gleichrichter	B80C800DIP	1
Diode 1A	1N 4001	1
grüne LED 2mA	LED 3MM 2MA GN	2
gelbe LED 2mA	LED 3MM 2MA GE	1
rote LED 2mA	LED 3MM 2MA RT	1
Kondensator 100nF	FOLIE MKS-2 100N	4
Kondensator 47µF	ELKO RAD 47/25V	2
Kondensator 33pF	KERKO 33P	2
Widerstand 1,2 KOhm	METALL 1,20K	4
Widerstand 10 KOhm	METALL 10K	1
Drosselspule 10µF	SMCC 0,10µ	1
Potentiometer	RT 10-L 47K	2
Miniaturtaster	TASTER 3301B	2
IC-Sockel	GS 28	1
Sockelleiste	SPL 20	2,5
Piezoschallwandler	SUMMER EPM121	1
Printstecker	PSS 254/2G	1
Standardquarz	3,686411-HC49U-S	1
Buchsenleiste	BL 1X20W 2,54	1
Buchsenleiste	BL 1X10W 2,54	2
Stiftleiste	SL 1X10W 2,54	2
Leiterplatte	myAVR-Board 2.x	1
USB-Programmer	Bestückte SMD Leiterplatte	1

Sicherheitshinweise

Grundsätzlich ist das myAVR Board nur zum Einsatz als Lern- und Experimentierplattform konzipiert. Es ist nicht vorgesehen und nicht dimensioniert zur Steuerung realer Anlagen. Bei vorschriftsmäßigem Anschluss und Betrieb treten keine lebensgefährlichen Spannungen auf. Beachten Sie trotzdem die Vorschriften, die beim Betrieb elektrischer Geräte und Anlagen Gültigkeit haben. Wir versichern, dass die Leiterplatte durch den Hersteller getestet wurde. Das Bauelementesortiment wurde gewissenhaft zusammengestellt und auf Vollzähligkeit überprüft. Für Fehler beim Bestücken der Leiterplatte leisten wir keinen Ersatz. Beschädigte Bauelemente senden wir Ihnen auf Anfrage zu. Für fehlerhaften und/oder vorschriftswidrigen Einsatz des Board übernehmen wir keine Garantie. Zum Anschluss des Boards an den PC ist ein Standard-USB-Kabel A-B zu verwenden. Der Einsatz anderer Kabel führt zu Fehlern bei der Programmierung und Kommunikation.

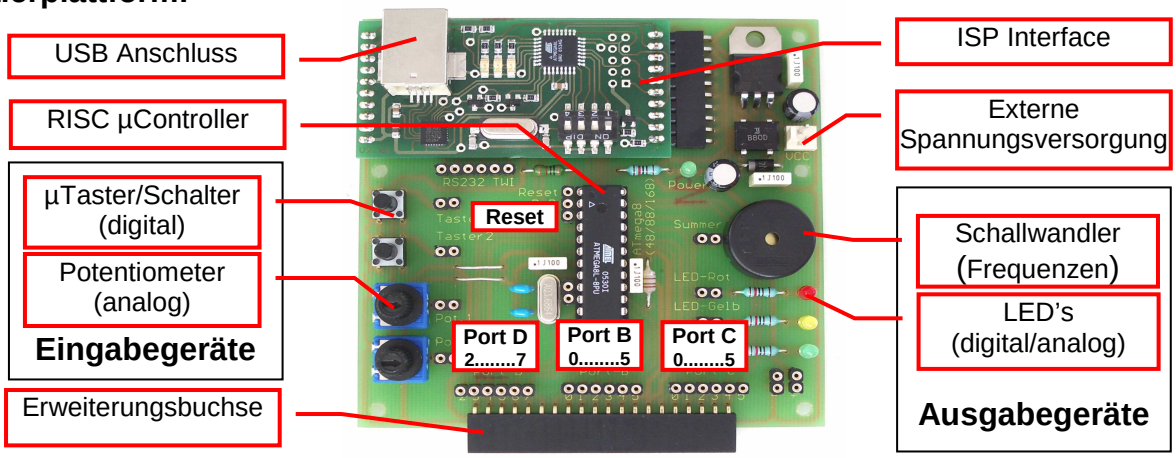
www.myAVR.de

Laser & Co. Solutions GmbH
Promenadenring 8
D-02708 Löbau

Email: hotline@myAVR.de
Tel: ++49 (0) 3585 470222
Fax: ++49 (0) 3585 470233

 ©Laser & Co. Solutions GmbH

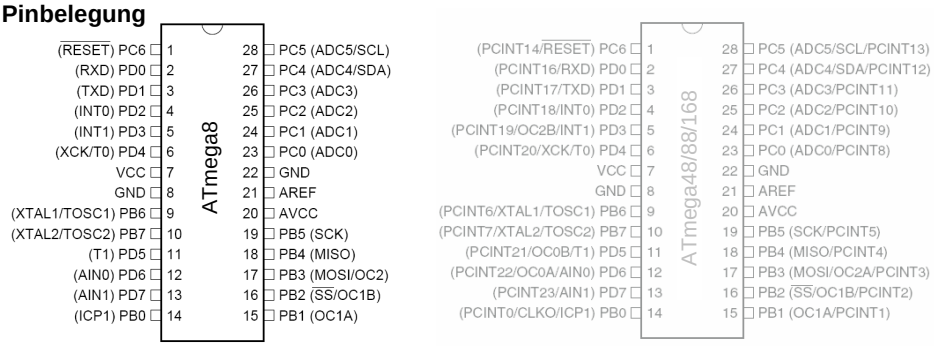
Experimentierplattform:



I/O Register

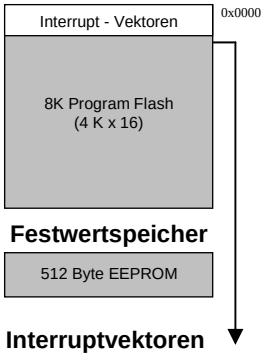
I/O	MEM	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x3F	0x5F	SREG	I	T	H	S	V	N	Z	C
0x3E	0x5E	SPH	–	–	–	–	–	SP10	SP9	SP8
0x3D	0x5D	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C	0x5B	Reserved								
0x3B	0x5B	GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE
0x3A	0x5A	GIFR	INTF1	INTF0	–	–	–	–	–	–
0x39	0x59	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0
0x38	0x58	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0
0x37	0x57	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
0x36	0x56	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
0x35	0x55	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
0x34	0x54	MCUCSR	–	–	–	–	WDRF	BORF	EXTRF	PORF
0x33	0x53	TCCR0	–	–	–	–	–	CS02	CS01	CS00
0x32	0x52	TCNT0	Timer/Counter0 (8 Bits)							
0x31	0x51	OSCCAL	Oscillator Calibration Register							
0x30	0x50	SFIOR	–	–	–	–	ACME	PUD	PSR2	PSR10
0x2F	0x4F	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
0x2E	0x4E	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0x2D	0x4D	TCNT1H	Timer/Counter1 – Counter Register High byte							
0x2C	0x4C	TCNT1L	Timer/Counter1 – Counter Register Low byte							
0x2B	0x4B	OCR1AH	Timer/Counter1 – Output Compare Register A High byte							
0x2A	0x4A	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte							
0x29	0x49	OCR1BH	Timer/Counter1 – Output Compare Register B High byte							
0x28	0x48	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte							
0x27	0x47	ICR1H	Timer/Counter1 – Input Capture Register High byte							
0x26	0x46	ICR1L	Timer/Counter1 – Input Capture Register Low byte							
0x25	0x45	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
0x24	0x44	TCNT2	Timer/Counter2 (8 Bits)							
0x23	0x43	OCR2	Timer/Counter2 Output Compare Register							
0x22	0x42	ASSR	-	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB
0x21	0x41	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0
0x20	0x40	UBRRH	URSEL	–	–	–	UBRR[11:8]	URSEL	–	–
		UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	URSEL	UMSEL	UPM1
0x1F	0x3F	EEARH	–	–	–	–	–	–	–	–
0x1E	0x3E	EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR7	EEAR6	EEAR5
0x1D	0x3D	EEDR	EEPROM Data Register							
0x1C	0x3C	EECR	–	–	–	–	EERIE	–	–	–
0x1B	0x3B	Reserved								
0x1A	0x3A	Reserved								
0x19	0x39	Reserved								
0x18	0x38	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x17	0x37	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x16	0x36	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x15	0x35	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x14	0x34	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x13	0x33	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x12	0x32	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x11	0x31	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x10	0x30	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x0F	0x2F	SPDR	SPI Data Register							
0x0E	0x2E	SPSR	SPIF	0x0E	(0x2E)	SPSR	SPIF	0x0E	(0x2E)	SPSR
0x0D	0x2D	SPCR	SPIE	0x0D	(0x2D)	SPCR	SPIE	0x0D	(0x2D)	SPCR
0x0C	0x2C	UDR	USART I/O Data Register							
0x0B	0x2B	UCSRA	RXC	0x0B	(0x2B)	UCSRA	RXC	0x0B	(0x2B)	UCSRA
0x0A	0x2A	UCSRB	RXCIE	0x0A	(0x2A)	UCSRB	RXCIE	0x0A	(0x2A)	UCSRB
0x09	0x29	UBRRL	USART Baud Rate Register Low byte							
0x08	0x28	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
0x07	0x27	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0
0x06	0x26	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
0x05	0x25	ADCH	ADC Data Register High byte							
0x04	0x24	ADCL	ADC Data Register Low byte							
0x03	0x23	TWDR	Two-wire Serial Interface Data Register							
0x02	0x22	TWAR	TWA6	TWA5	TWA4	TWA3	TWGCE	TWA2	TWA1	TWA0
0x01	0x21	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
0x00	0x20	TWBR	Two-wire Serial Interface Bit Rate Register 168							

Pinbelegung

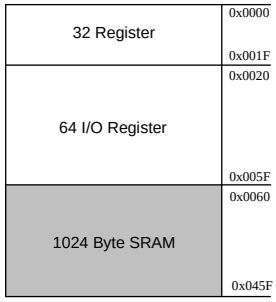


Speicheraufbau

Programmspeicher



Arbeitsspeicher



Festwertspeicher

Interruptvektoren

Vector No.	Program Adress	Source	Interrupt Definition
1	0x000	RESET	External Pin,Power-on Reset,Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare MatchA
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare MatchB
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI; STC	Serial Transfer Complete
12	0x00B	USART , RXC	USART, Rx Complete
13	0x00C	USART , UDRE	USART, Data Register Empty
14	0x00D	USART , TXC	USART, TX Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comperator
18	0x011	TWI	Two-wire serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Arbeitsregister

Register			Adresse	Bemerkung
R0			0x00	siehe LPM
R1			0x01	**KEIN** ADIW, SUBI, SUBIW, ANDI, ORI, CPI, LDI,
...			...	
R15			0x0F	
R16			0x10	keine Einschränkungen
...			...	
R26	XL	X	0x1A	siehe LD/ST
R27	XH		0x1B	
R28	YL	Y	0x1C	siehe LD/ST
R29	YH		0x1D	
R30	ZL	Z	0x1E	siehe LD/ST
R31	ZH		0x1F	

Portfunktionen

Name	Port	Funktion
Externe Interrupts	D2-3	INT0-1
Analog-Digital-Wandler	C0-5	ADC0-5
Analog-Komparator	D6-7	AIN0-1
myAVR LCD	D2	R/S
	D3	Enable
	D4-5	DB4-7
	B0 (optional)	R/W
	B1 (optional)	Backlight
TWI / I²C	C4, C5	SDA, SCL
SPI-Bus AVR-ISP	B3	MOSI
	B4	MISO
	B5	SCK
	B2 (optional)	SS
UART (RS232)	D0, D1	RxD, TxD
	D4 (für Sync.Mode)	XCK
Reset	C6	RST
Quarz/Resonator/Takt	B6, B7	XTAL1, XTAL2
Timer/Counter-Output (PWM, Waveform)	B1, B2, B3	OC1A, OC1B, OC2
Timer/Counter Input-Clock	D4, D5	T0, T1
Timer/Counter InputCapture	B0	ICP1

© Laser & Co. Solutions GmbH

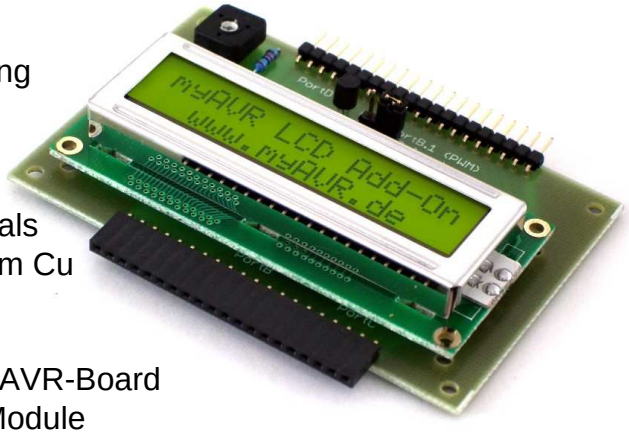
Befehlssatz

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	K Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd − 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
ULSU	Rd, Rr	Fractional Multiply Signed wiTh Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CP	Rd,Rr	Compare	Rd − Rr	Z,N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z,N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd − K	Z,N,V,C,H	1
CPSE	Rd,Rr	Compare, Skip if Equal if (Rd = Rr)	PC ← PC + 2 or 3	None	1/2/3
SBRC	Rr, b	Skip if Bit in Register Cleared if (Rr(b)=0)	PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set if (Rr(b)=1)	PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM (Data Mem.)	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM (Data Mem.)	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1 SREG(s)		1
BCLR	s	Flag Clear	SREG(s) ← 0 SREG(s)		1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		WatchdogReset	(see specific descr. for WDR/timer)	None	1

```
;+--------------------------------------------------------------------------------
;| Titel       : Beispiel IN/OUT für das myAVR-Board
;+--------------------------------------------------------------------------------
;| Funktion    : Solange Taster 1 gedrückt ist, wird eine LED eingeschaltet
;| Schaltung   : PORTD2=Taster1, PORTB0=LED-Rot
;+--------------------------------------------------------------------------------
.include    "AVR.H"
;--- Reset and Interrupt vector  ----  ;VNr.  Beschreibung -----
            rjmp    main                ;1     POWER ON RESET
            reti                        ;2     Int0-Interrupt
            reti                        ;3     Int1-Interrupt
            reti                        ;4     TC2 Compare Match
            reti                        ;5     TC2 Overflow
            reti                        ;6     TC1 Capture
            reti                        ;7     TC1 Compare Match A
            reti                        ;8     TC1 Compare Match B
            reti                        ;9     TC1 Overflow
            reti                        ;10    TC0 Overflow
            reti                        ;11    SPI, STC Serial Transfer Complete
            reti                        ;12    UART Rx Complete
            reti                        ;13    UART Data Register Empty
            reti                        ;14    UART Tx Complete
            reti                        ;15    ADC Conversion Complete
            reti                        ;16    EEPROM Ready
            reti                        ;17    Analog Comparator
            reti                        ;18    TWI (IC) Serial Interface
            reti                        ;19    Store Program Memory Ready
;--- Start, Power ON, Reset ------------------------------------------------
main:       ldi     r16,hi8(RAMEND)     ; Stack Initialisierung
            out     SPH,r16
            ldi     r16,lo8(RAMEND)
            out     SPL,r16
            cbi     DDRD,2              ; PORTD2 auf Eingang
            sbi     PORTD,2             ; PORTD2 Pullup
            sbi     DDRB,0              ; PORTB0 auf Ausgang
;--------------------------------------------------------------------------------
mainloop:   ldi     r16,0               ; Wert bei Taste nicht gedrückt
            sbis    PIND,2              ; Taste auswerten
            ldi     r16,1               ; Wert bei Taste gedrückt
            out     PORTB,r16           ; LED an/aus
            rjmp    mainloop
;--------------------------------------------------------------------------------
```


© Laser & Co. Solutions GmbH

Eigenschaften



- 16x2 LC-Display mit Hintergrundbeleuchtung
- Industriestandard-Controller
- Kontrastregler
- Jumper für das Schalten der Hintergrundbeleuchtung und des RW-Signals
- Leiterplattenmaterial: FR4, 1,5 mm; 0,35 µm Cu
- Industriefertigung, gebohrt, verzinnt, Lötstoppmaske, Bestückungsdruck, robust
- Steckerleiste für den Anschluss an das myAVR-Board
- Buchsenleiste für den Anschluss weiterer Module

Allgemeine Beschreibung

Das myAVR LCD Add-On ist ein anschlussfertiges LCD-Modul für das myAVR-Board 1 LPT oder das myAVR-Board 2 USB. Es ist mit einem hochwertigen LC-Display mit 2 Zeilen je 16 Zeichen ausgestattet. Die Hintergrundbeleuchtung kann per Jumper geschaltet oder per Software vom Controller geregelt werden. Zusätzlich verfügt das Board über eine Kontrastreglung und Anschlüsse für das myAVR-Board und weitere Add-Ons. Das LCD Add-On ist robust und auf die myAVR Produktreihe abgestimmt.

Technische Daten

Maximalwerte

Maximalstrom:	150 mA (bei eingeschalteter Hintergrundbeleuchtung)
Maximalspannung:	5,5 V
Minimalspannung:	4,5 V
Lagertemperatur:	-10°C bis +40°C

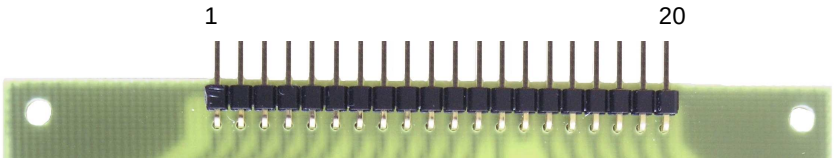
Beachte: *Beim Betrieb von zusätzlichen Verbrauchern am myAVR Board ist auf die Gesamtleistung der Stromversorgung (maximal 1000 mA nur mit Kühlkörper) zu achten. Die Betriebstemperatur des Spannungsreglers (7806) ist während des Betriebes zu überprüfen und gegebenenfalls ein Kühlkörper nachzurüsten. Die Belastung des Spannungsreglers kann vor allem durch den Anschluss einer geeigneten Spannungsquelle verringert werden. Umso höher die Spannung an der Spannungsquelle ist, umso mehr Wärme muss vom Spannungsregler abgeführt werden. Die Spannungsquelle darf 12V nicht überschreiten. Das Optimum ist ein 9V Gleichspannung. Diese erhalten Sie zum Beispiel aus einem stabilisierten 9V Netzteil mit max. 1000mA.*

Betriebsdaten

Betriebsstrom:	ca. 120 mA	(bei eingeschalteter Hintergrundbeleuchtung)
Betriebsstrom:	ca. 5 mA	(ohne Hintergrundbeleuchtung)
Betriebsspannung:	typisch 4,8 – 5,3 V	
Betriebstemperatur:	0°C bis +30°C	

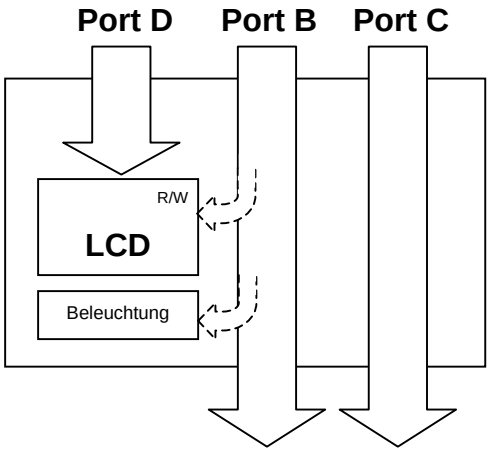
Schnittstellendaten

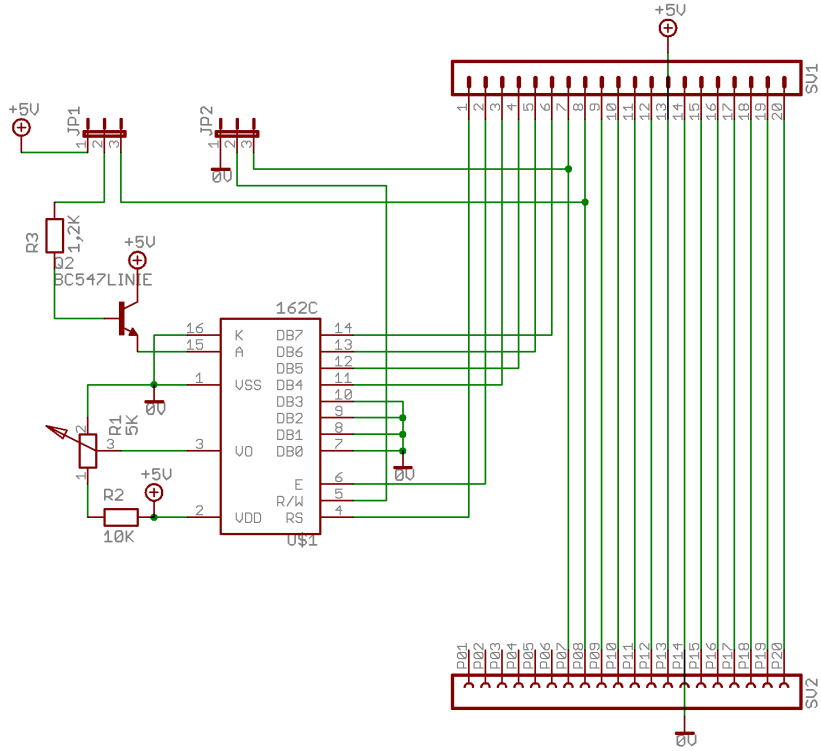
Steckerleiste für den Anschluss an das myAVR-Board



PIN-Belegung:

1 = Port D.2	7 = Port B.0	13 = 5 V	15 = Port C.0
2 = Port D.3	8 = Port B.1	14 = Masse	16 = Port C.1
3 = Port D.4	9 = Port B.2		17 = Port C.2
4 = Port D.5	10 = Port B.3		18 = Port C.3
5 = Port D.6	11 = Port B.4		19 = Port C.4
6 = Port D.7	12 = Port B.5		20 = Port C.5



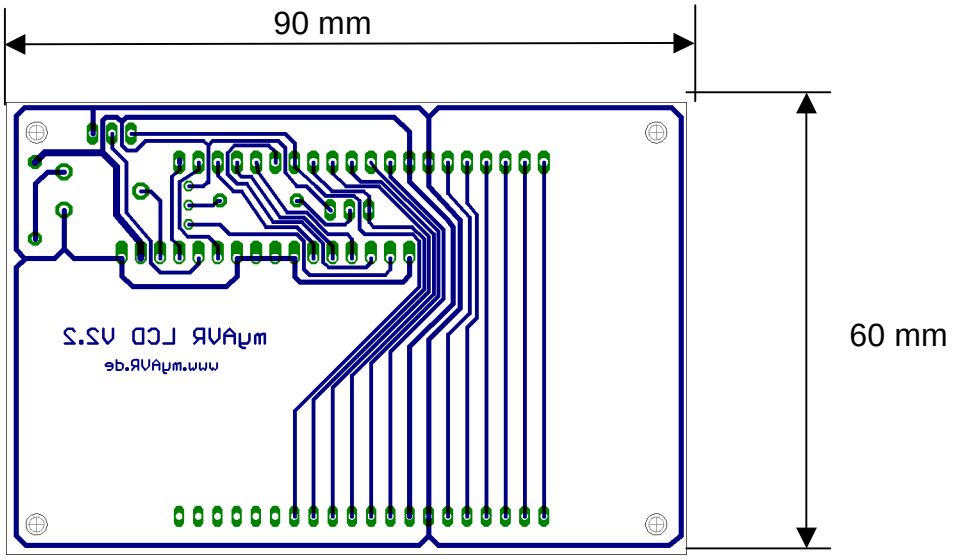


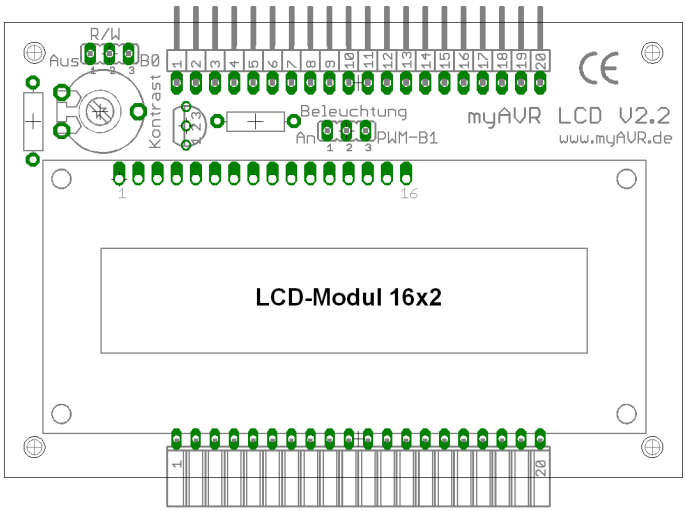
LCD Belegung (4 Bit Modus)

1 = Port D.2	LCD PIN 4	RS = Enable
2 = Port D.3	LCD PIN 6	E = Enable
3 = Port D.4	LCD PIN 11	DB4 = Datenbit 4
4 = Port D.5	LCD PIN 12	DB5 = Datenbit 5
5 = Port D.6	LCD PIN 13	DB6 = Datenbit 6
6 = Port D.7	LCD PIN 14	DB7 = Datenbit 7
7 = Port B.0 (Jumper R/W)	LCD PIN 5	R/W =Read/Write (oder fest auf Low)
8 = Port B.1 (Jumper PWM)	LCD PIN 15	Hintergrundbeleuchtung (Anode)
Kontrastreglung	LCD PIN 3	Vo
13 = 5V	LCD PIN 1	Vss
14 = Masse	LCD PIN 2	Vdd
	LCD PIN 7	DB0 = Datenbit 0
	LCD PIN 8	DB1 = Datenbit 1
	LCD PIN 9	DB2 = Datenbit 2
	LCD PIN 10	DB3 = Datenbit 3
	LCD PIN 15	Hintergrundbeleuchtung (Kathode)

Mechanische Daten

Material:	FR4; 0,35 µm Cu
Abmaße (L x B):	90 mm x 60 mm
Dicke:	1,5 mm
Rastermaß:	2,54 mm
Gewicht:	ca. 55 g



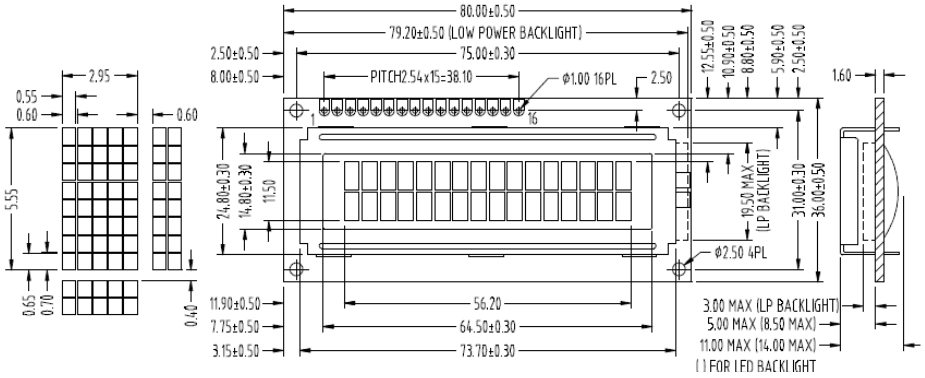


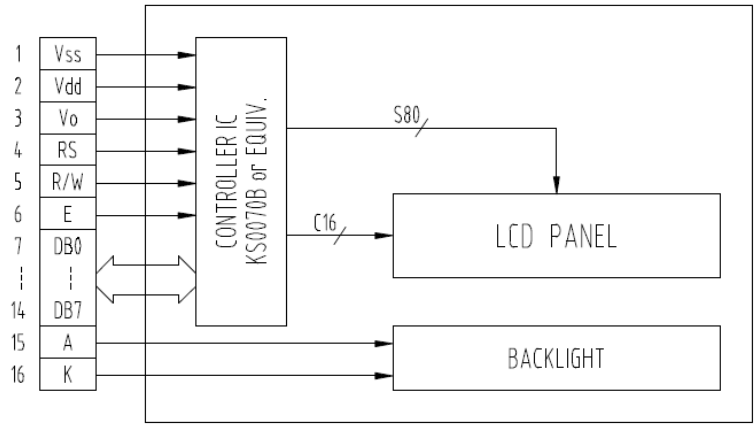
Stückliste:

LCD Add-On Leiterplatte	1
LCD 16x2 mit BL	1
Buchsenleiste 1x20W	1
Steckerleiste 1x20W	1
Jumper 1x3	2
Potentiometer 5K	1
Widerstand 10K	1
Widerstand 220	1
Transistor	1

Displaydaten:

Item	Contents	Unit
LCD type	TN / STN / FSTN	---
LCD duty	1/16	---
LCD bias	1/5	---
Viewing direction	6 / 12	o'clock
Module size (W×H×T)	80.0 × 36.0 × 11.0 MAX (14.0 MAX W/LED BACKLIGHT)	mm
Viewing area (W×H)	64.5 × 14.8	mm
Number of characters (characters×lines)	16 × 2	---
Character matrix (W×H)	5 × 8	dots
Character size (W×H)	2.95 × 5.55	mm
Dot size (W×H)	0.55 × 0.65	mm
Dot pitch (W×H)	0.60 × 0.70	mm





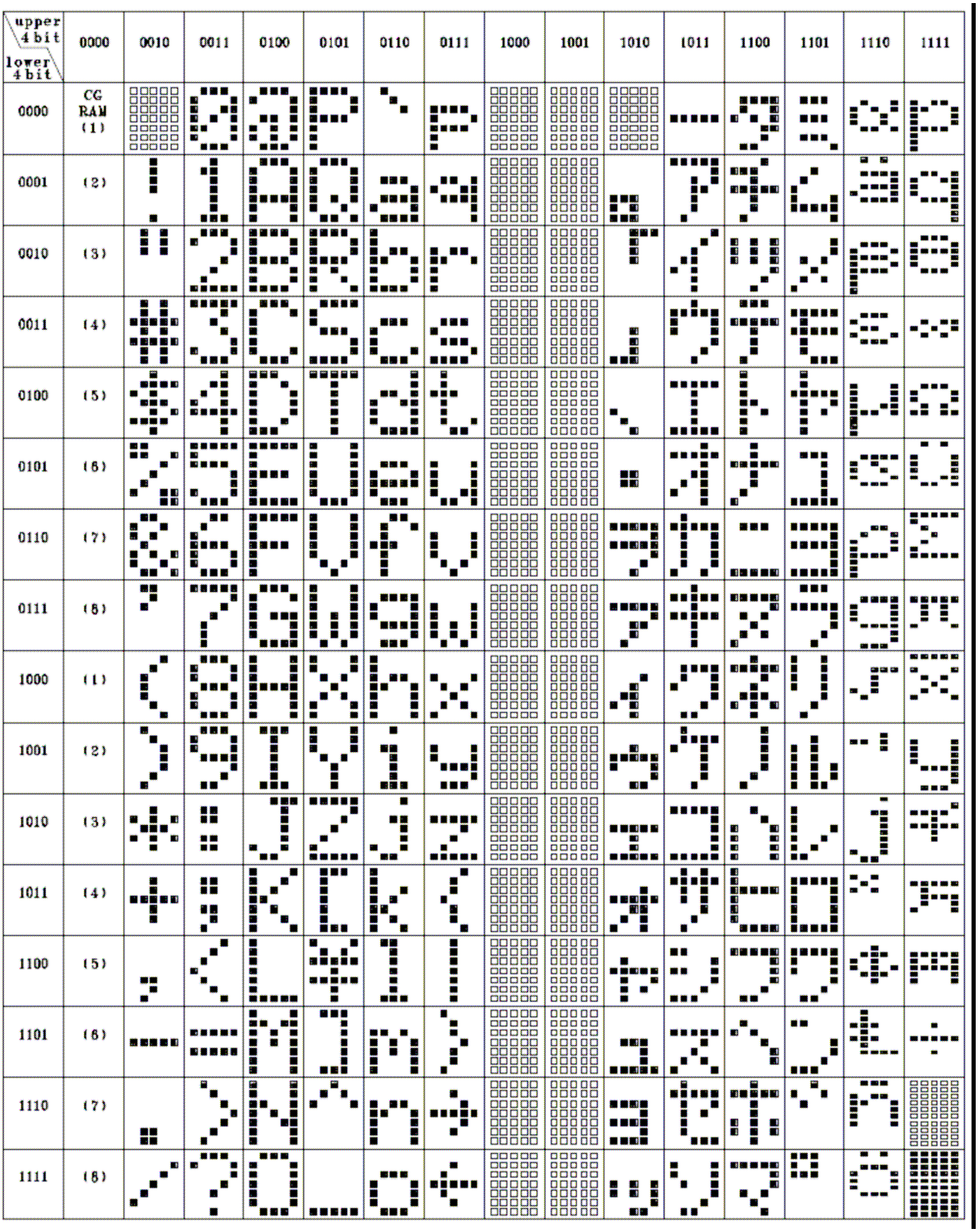
Pin NO.	Symbol	Level	Description
1	VSS	0V	Ground
2	VDD	5.0V	Supply voltage for logic
3	VO	---	Input voltage for LCD
4	RS	H/L	H : Data, L : Instruction code
5	R/W	H/L	H : Read mode, L : Write mode
6	E	H, H → L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	BLA	---	Back light anode
16	BLK	---	Back light cathode

Displaykommandos:

Command	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	Execution Time (f_{osc} = 250kHz)	Remark
DISPLAY CLEAR	L	L	L	L	L	L	L	L	L	H	1.64ms	
RETURN HOME	L	L	L	L	L	L	L	L	H	X	1.64ms	Cursor move to first digit
ENTRY MODE SET	L	L	L	L	L	L	L	H	I/D	SH	42µs	• I/D : Set cursor move direction I/D: H = Increase; L = Decrease • SH : Specifies shift of display SH: H = Display is shifted; L = Display is not shifted
DISPLAY ON/OFF	L	L	L	L	L	L	H	D	C	B	42µs	• Display D: H = Display on; L = Display off • Cursor C: H = Cursor on; L = Cursor off • Blinking B: H = Blinking on; L = Blinking off
SHIFT	L	L	L	L	L	H	S/C	R/L	X	X	42µs	S/C: H = Display shift; L = Cursor move R/L: H = Right shift; L = Left shift
SET FUNCTION	L	L	L	L	H	DL	N	F	X	X	42µs	DL: H = 8 bits interface; L = 4 bits interface N: H = 2 line display; L = 1 line display F: H = 5 X 10 dots; L = 5 X 7 dots
SET CG RAM ADDRESS	L	L	L	H	CG RAM address (corresponds to cursor address)						42µs	CG RAM Data is sent and received after this setting
SET DD RAM ADDRESS	L	L	H	DD RAM address							42µs	DD RAM Data is sent and received after this setting
READ BUSY FLAG & ADDRESS	L	H	BF	Address Counter used for both DD & CG RAM address							0µs	BF: H = Busy; L = Ready – Reads BF indication internal operating is being performed – Reads address counter contents
WRITE DATA	H	L	Write Data								46µs	Write data into DD or CG RAM
READ DATA	H	H	Read Data								46µs	Read data from DD or CG RAM

X : Don't care

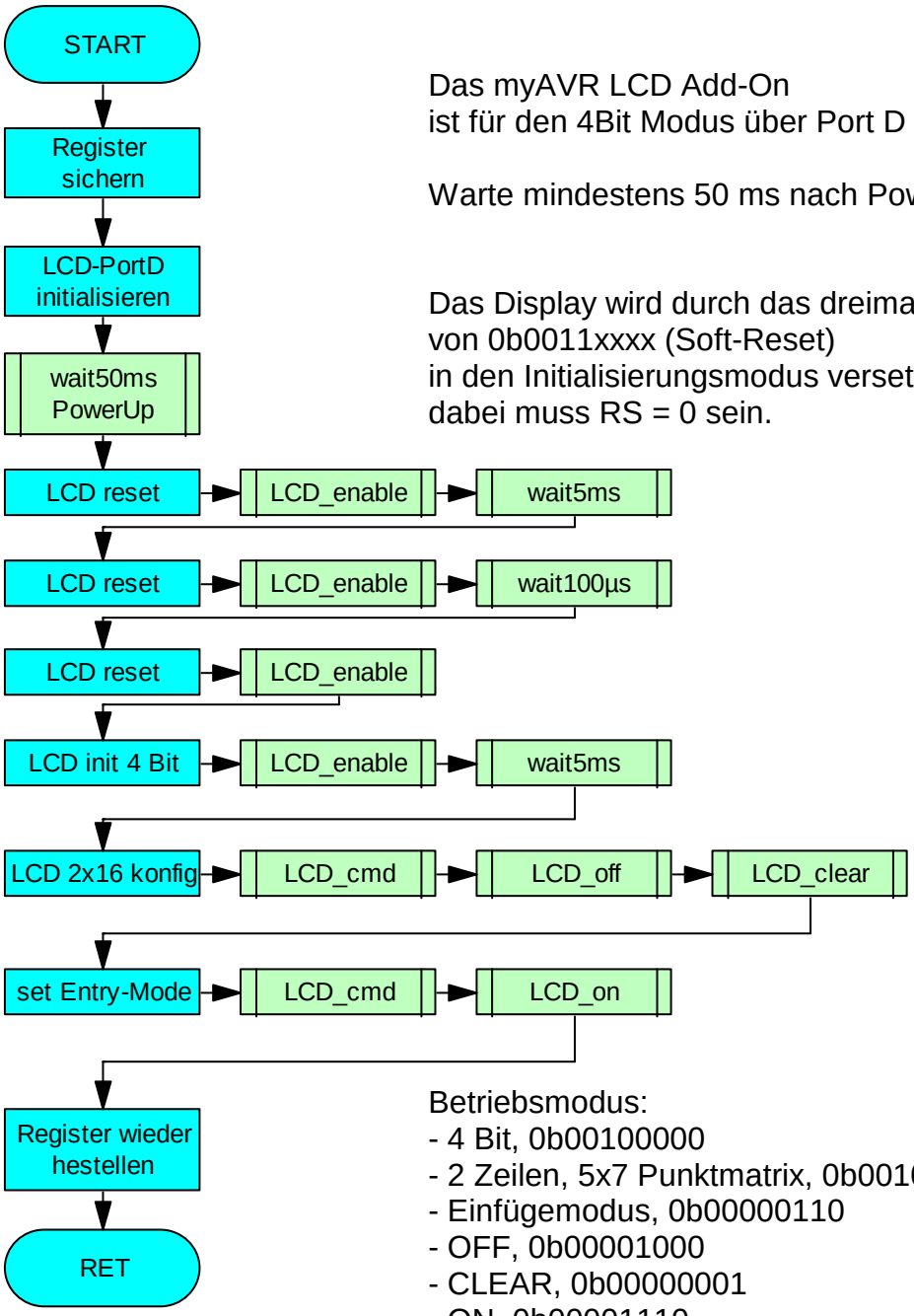
Displayzeichensatz:



Display-Adressorganisation:

Characters	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
First line	00H	01H	02H	03H	04H	05H	06H	07H	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
Second line	40H	41H	42H	43H	44H	45H	46H	47H	48H	49H	4AH	4BH	4CH	4DH	4EH	4FH

Programmierung des Displays:
Beispiele und Vorlagen für die Programmierung des myAVR LCD Add-On finden Sie im Downloadbereich von www.myAVR.de.



Das myAVR LCD Add-On
ist für den 4Bit Modus über Port D konfiguriert.

Warte mindestens 50 ms nach Power-ON.

Das Display wird durch das dreimalige senden
von 0b0011xxxx (Soft-Reset)
in den Initialisierungsmodus versetzt,
dabei muss RS = 0 sein.

Betriebsmodus:
- 4 Bit, 0b00100000
- 2 Zeilen, 5x7 Punktmatrix, 0b00101000
- Einfügemodus, 0b00000110
- OFF, 0b00001000
- CLEAR, 0b00000001
- ON, 0b00001110

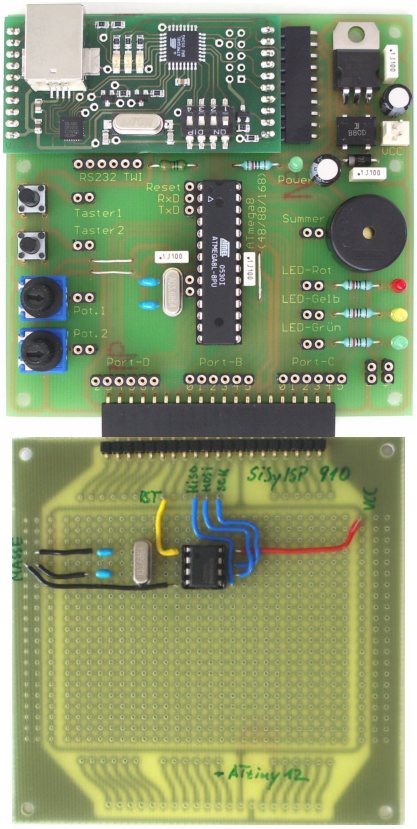
BASCOM Beispiel:

```
$regfile = "m8def.dat"
$crystal = 3686400
Config Lcdpin = Pin, Db4 = Portd.4, Db5 = Portd.5, Db6 = Portd.6, Db7 = Portd.7,
E = Portd.3, Rs = Portd.2
Config Lcd = 16 * 2
Config Portd = Output
Do
  Waitms 1000
  Cls
  Lcd "myAVR mit LCD"
Loop
End
```

Dipl. Ing. Toralf Riedel
Dipl. Ing. Päd. Alexander Huwaldt

myTinyProg

Projektbeschreibung zum AVR910 Tiny Programmer für das myAVR Board



Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Die Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind die Autoren dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Dokument erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden.

4. Auflage: Mai 2006

© Laser & Co. Solutions GmbH
www.laser-co.de
www.myavr.de
info@myavr.de.de
Tel: ++49 (0) 3585 470 222
Fax: ++49 (0) 3585 470 233

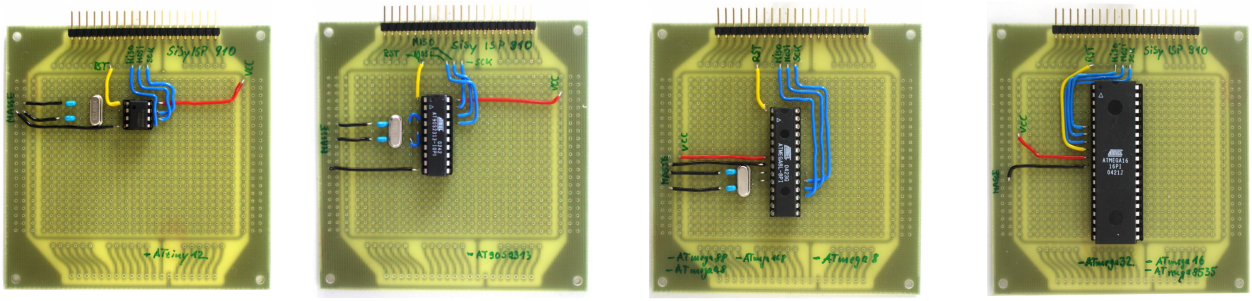
Inhalt

1 Einleitung 5
2 Grundlagen 6
2.1 Atmel Application Note 910 7
3 Hardware myAVR-Board und myTinyProg 9
3.1 Hardwareüberblick 9
3.2 Schaltplan myTinyProg 10
3.3 Stückliste 10
3.4 Realisierung auf einer myAVR-Laborkarte 11
4 myTinyProg Firmware 12
4.1 Beschreibung der Firmware 12
4.2 Brennen der Firmware 12
4.3 Brennen (Programmieren) des Tiny's 13
4.4 myTinyProg mit SiSy AVR 13
4.5 myTinyProg mit myAVR Workpad PLUS 13
5 Quellenverzeichnis 14

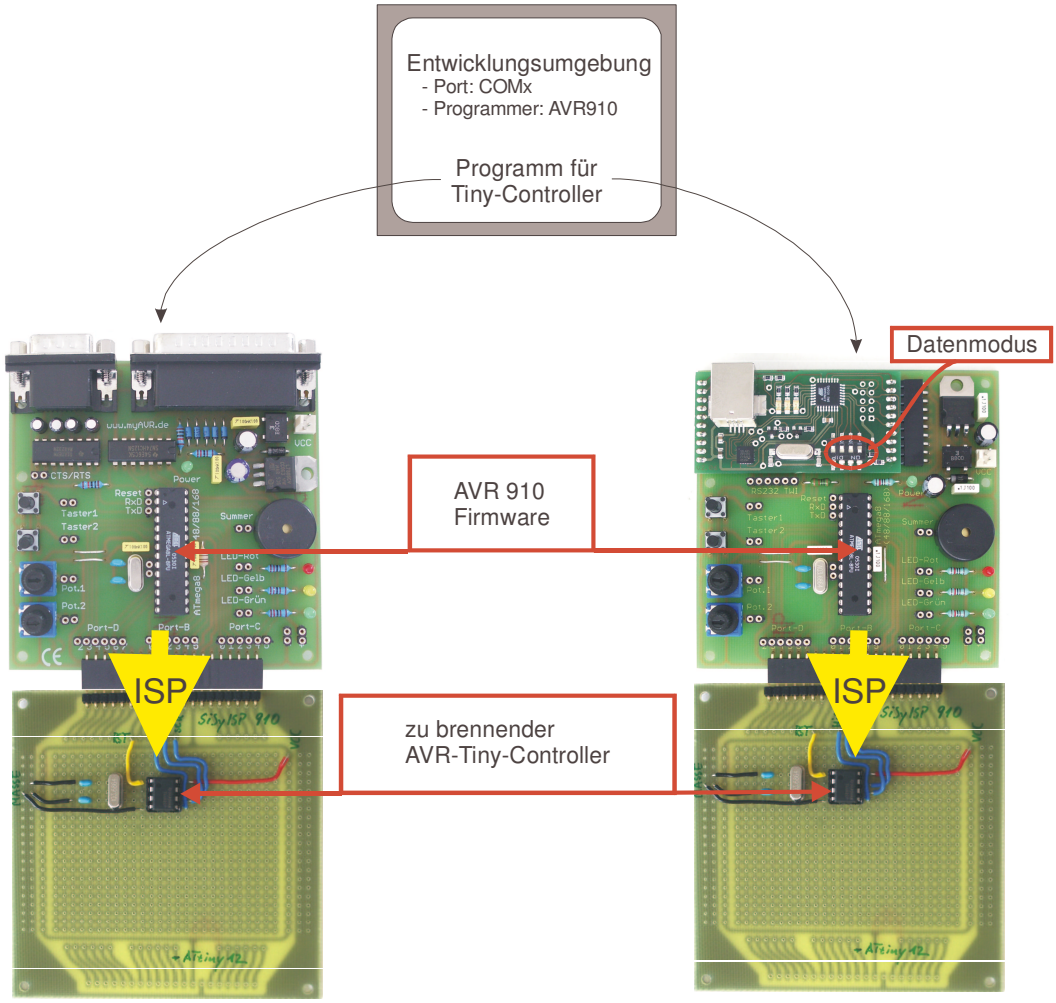
 © Laser & Co. Solutions GmbH

1 Einleitung

Das vorliegende Projekt *myTinyProg* beschreibt eine Möglichkeit, das Lern- und Experimentiersystem myAVR als einen komfortablen seriellen AVR910-Programmer für AVR-Tiny Controller zu nutzen. Die Basis ist dafür ist eine Laborkarte A und wenige Bauteile, wie ein DIP8-Sockel, ein Quarz und zwei Keramikkondensatoren. Darüber hinaus soll das Projekt Anregungen geben, diesen Einsatzfall zum Beispiel auch für Controller der AVR-Mega-Serie weiter zu entwickeln.



Dieses Projekt ist geeignet für das myAVR Board 1 LPT und das myAVR Board 2 USB.



Wir wünschen Ihnen viel Erfolg und Spaß beim Lernen und Experimentieren.

Ihr myAVR-Team

2 Grundlagen

Für das Brennen (Programmieren, Flashen) des fertigen Mikrocontrollerprogramms (*.HEX oder *.BIN) gibt es grundsätzlich zwei Möglichkeiten:

- Zum Einen kann man ein Programmiergerät verwenden, in das man den Chip einsetzt und programmiert. Dazu muss das Programmiergerät zum Beispiel über die serielle Schnittstelle oder USB an den PC angeschlossen werden. Der Mikrocontroller ist aus dem Zielsystem zu entfernen und auf dem entsprechenden Sockel des Programmiergerätes zu stecken. Dann kann das Programm in den FLASH-Speicher des Controllers übertragen werden. War dieser Vorgang erfolgreich, kann der Controller aus dem Programmiergerät entnommen und wieder in das Zielsystem eingebaut werden.

- Eine weitere Lösung für die Programmierung des Mikrocontrollers ist das sogenannte „In System Programming“ (ISP). Dabei muss der Controller nicht aus dem Zielsystem ausgebaut werden, sondern kann direkt im System programmiert werden. Dafür muss das Zielsystem jedoch eine ISP-Schnittstelle bereitstellen. Mit einer zusätzlichen Hardware, dem sogenannten ISP-Programmer, der an den LPT-Port, die COM-Schnittstelle oder den USB-Port angeschlossen wird, kann aus der Entwicklungsumgebung heraus das Programm direkt in das Zielsystem übertragen werden.

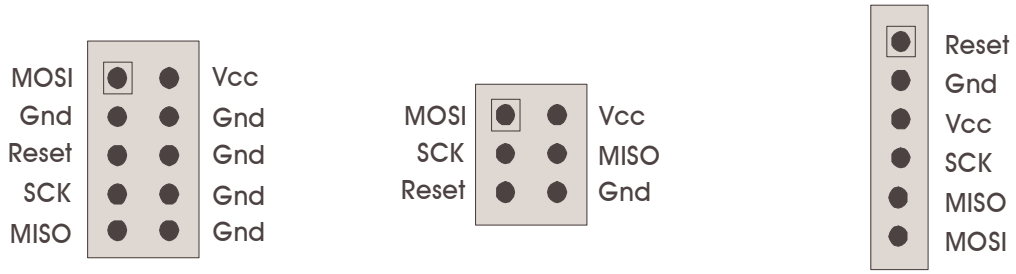


Abbildung: ISP- Anschlüsse, ISP- Standards, Atmel 10-polig, 6-polig, TwinAVR

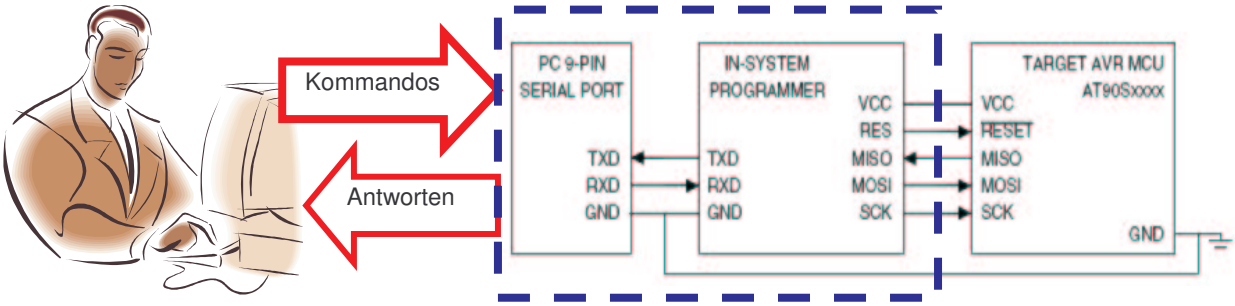
Die folgende Tabelle zeigt die Bedeutungen der einzelnen Signalleitungen der ISP-Schnittstelle. Diese sind die Leitungen des SPI (Serial Processor Interface).

Pin	Name	Comment
SCK	Serial Clock	Programming clock, generated by the In-System programmer (master)
MOSI	Master Out – Slave In	Communication line from In-System programmer (master) to target AVR being programmed (slave)
MISO	Master In – Slave Out	Communication line from target AVR (slave) to In-System programmer (master)
GND	Common Ground	The two systems must share the same common ground
RESET	Target AVR MCU Reset	To enable In-System programming, the target AVR Reset must be kept active. To simplify this, the In-System programmer should control the target AVR Reset
V_{CC}	Target Power	To allow simple programming of targets operating at any voltage, the In-System programmer can draw power from the target. Alternatively, the target can have power supplied through the In-System programming connector for the duration of the programming cycle

Für die Programmer-Hardware gibt es eine Reihe von konkreten Lösungen. Atmel selbst stellt im Application Note AVR910 eine Programmerlösung vor, die inzwischen von den meisten Programmierumgebungen unterstützt wird.

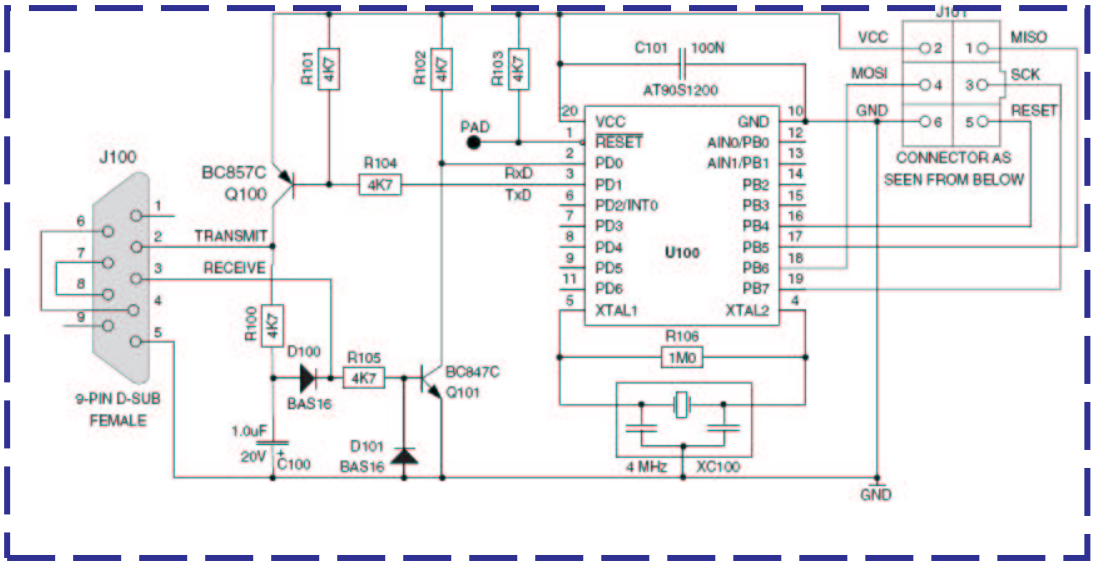
2.1 Atmel Application Note 910

Atmels Application Note AVR910 stellt zwei Aspekte einer einfachen Programmerlösung dar. Zum Einen eine Hardwarelösung und zum Anderen ein Softwareprotokoll. Die Hardware basiert auf einem RS232 Interface, einem AVR-Controller und dem eigentlichen ISP-Interface. Die Programmierumgebung sendet über den COM-Port an den Programmer Kommandos entsprechend dem AVR910 Protokoll. Diese werden durch die Firmware in ISP-Kommandos (ISP-Protokoll) umgesetzt. Da der AT90S1200 über kein SPI (Serial Processor Interface) verfügt, ist in der Firmware eine Software lösung dafür zu finden (Soft-SPI).



AVR910 Hardware

Die originale Hardwarelösung sieht im Detail wie folgt aus:
Sie besitzt eine einfache Pegelanpassung für die COM-Schnittstelle des PC (RS232) und einen AT90S1200. Die Firmware für den AT90S1200, welche das AVR910 Protokoll realisiert, stellt Atmel auch bereit.



Ausgehend von diesem Schema liegt der Schluss nahe, dass ein myAVR Board ja ebenfalls über diese Komponenten verfügt.

- Die Pegelanpassung für die serielle Schnittstelle übernimmt beim myAVR Board 1 LPT ein RS232 Treiber und für die Firmware haben wir einen ATmega8, der bereits über ein SPI verfügt.
- Bei dem myAVR Board 2 USB erfolgt die serielle Verbindung über den bereits installierten COM-Port-Treiber.

Also beste Vorraussetzungen, daraus einen AVR910 Programmer zu machen ;-)

AVR910 Firmware

Wichtiger als die eigentliche AVR910 Hardware ist die Softwarekomponente, die das Kommunikationsprotokoll mit der Entwicklungsumgebung realisiert. Die folgende Tabelle zeigt die AVR910 Kommandos, die per serieller Schnittstelle vom Brennprogramm an den Programmer gesendet werden.

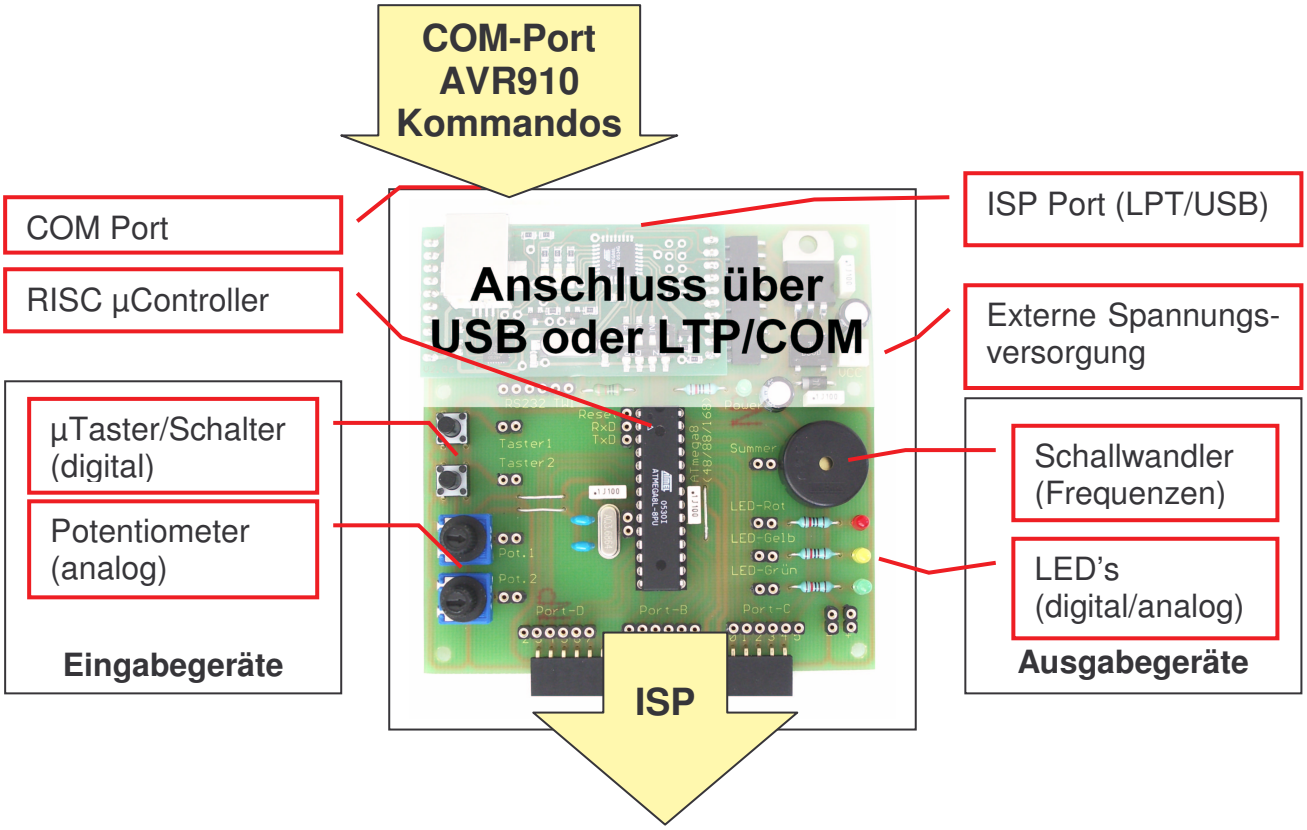
Kommando	Code	Beschreibung	wait
Programming Enable	**0xAC 0x53** ** **	Nach Power UP und RESET Low muss als erstes dieses Kommando gesendet werden, danach akzeptiert der Controller Programmier-Kommandos	
Chip Erase	**0xAC 0x8*** ** **	Bevor der Programmspeicher neu programmiert werden kann, ist dieser erst zu löschen. Dies bewirkt, dass jedes Byte mit 0xFF (alle Bits High) überschrieben wird. Der Programmiervorgang setzt dann nur noch die Low-Bits.	x
Write FLASH Low Write FLASH Higth	**0x60** adrh adrl byte **0x68** adrh adrl byte	Jeder Maschinenbefehl des AVR besteht aus 2 Byte (16 Bit Worte). Diese sind einzeln zu übertragen.	x
Read FLASH Low Write FLASH Higth	**0x20** adrh adrl ** **0x28** adrh adrl **	Jeder Maschinenbefehl des AVR besteht aus 2 Byte. Diese können nur einzeln gelesen werden.	
Write EEPROM	**0xC0** adrh adrl byte	Schreibt ein Byte in den EEPROM. Dieser ist byteorientiert.	x
Read EEPROM	**0xA0** adrh adrl **	Liest ein Byte aus dem EEPROM. Dieser ist byteorientiert.	
Read Vendor Code	**0x30** ** 0x00 **	Liest den Herstellercode aus. Atmel signiert mit 0x1e.	
Read Family Code	**0x30** ** 0x01 **	Liest den Gruppencode aus. Der Mega8 liefert zum Beispiel 0x07. Darin ist Controllerfamilie und FLASH-Size codiert.	
Read Part Number	**0x30** ** 0x02 **	Liest den Teilcode aus. Der Mega8 liefert zum Beispiel 0x93. Aus Herstellercode, Gruppencode und Teilcode ergibt sich die Controllersignatur Mega8 = 0x1e9307	

Einige Kommandos erfordern controllerspezifische Ausführungszeiten. Diese können den Datenblättern der Controller entnommen werden.

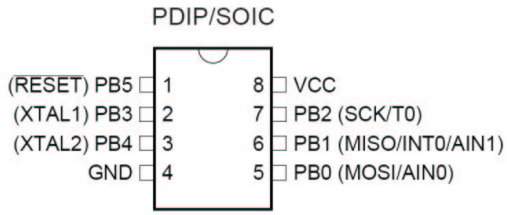
3 Hardware myAVR Board und myTinyProg

3.1 Hardwareüberblick

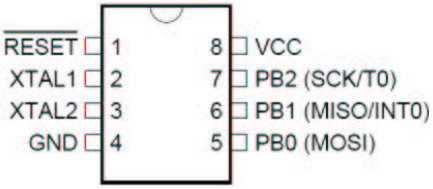
Das myAVR Board besitzt folgende Struktur:
LPT- oder USB-Programmer, RS232 Interface, ATmega8 Controller, Peripherie, Erweiterungsbuchse. Bei diesem Projekt soll das Board als AVR910 Programmer arbeiten.



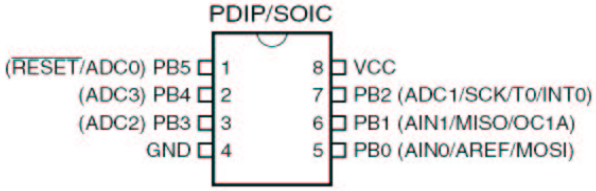
ATtiny12:



AT90S2323:



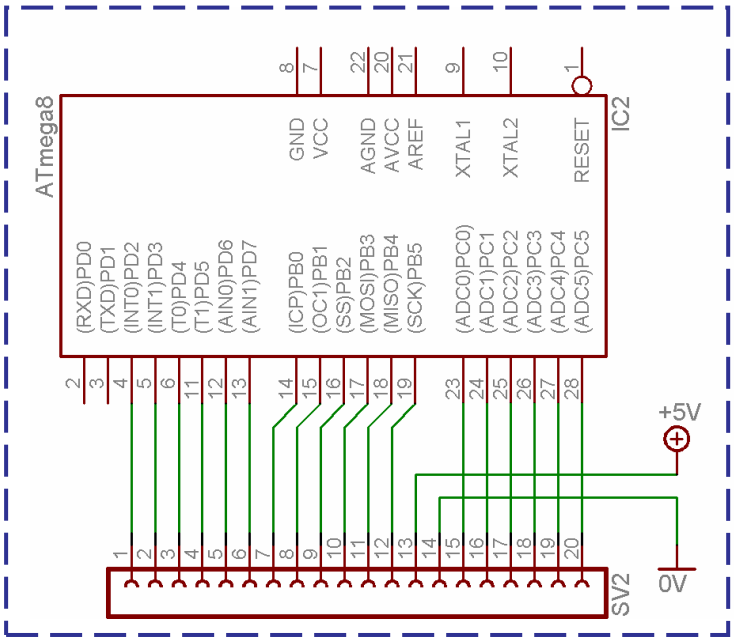
ATtiny15:



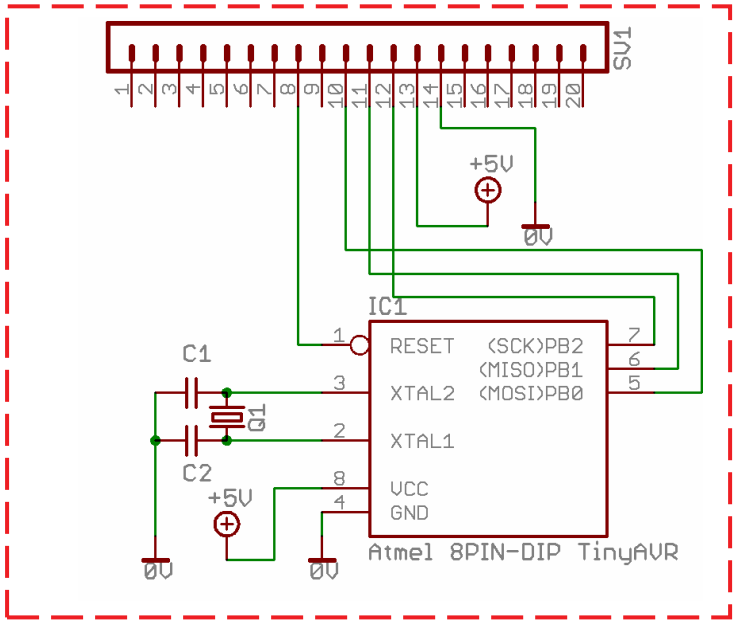
3.2 Schaltplan myTinyProg

Die Schaltung soll die Programmierung der Atmel Controller AT90S2323, ATtiny12, 13 und 15 ermöglichen. Dabei sind die ISP Leitungen entsprechend der PIN-Belegung der Controller und der Erweiterungsbuchse des myAVR-Boards zu verbinden. Die SPI Signale sind am Mega8 auf Port B 3, 4 und 5 verfügbar.

myAVR-Board (Auszug)



Laborkarte A

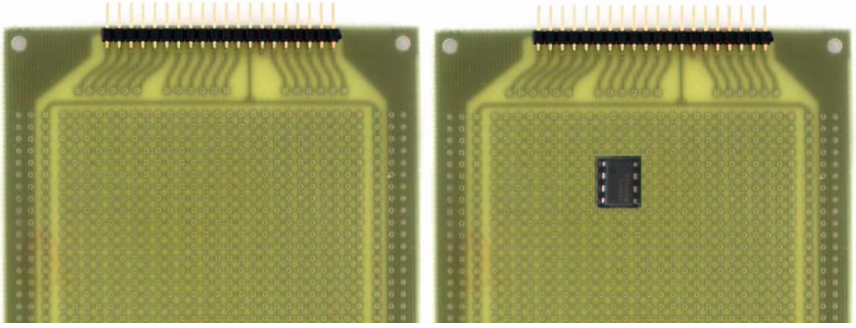


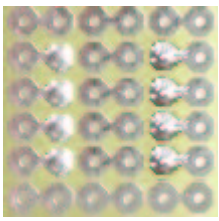
3.3 Stückliste

IC1	ATtiny12/13/15 DIP	1	Stück
J1	IC Sockel 8 DIP	1	Stück
C1, C2	KERKO 33p	2	Stück
Q1	Quarz 3,6864 MHz	1	Stück
X1	Laborkarte A (Platine)	1	Stück
SV1	Stiftleiste 20polig	1	Stück
	Schaltdraht	ca. 20 Zentimeter (Farbsortiment)	

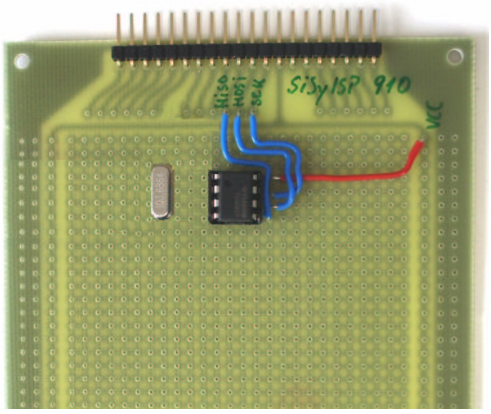
3.4 Realisierung auf einer myAVR Laborkarte A

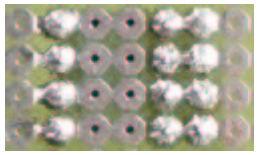
1. Löten Sie Buchsenleiste und IC Sockel auf die Laborkarte. Achten Sie darauf, dass die paarigen Lötaugen der Laborkarte einen einfachen Anschluss gewährleisten.



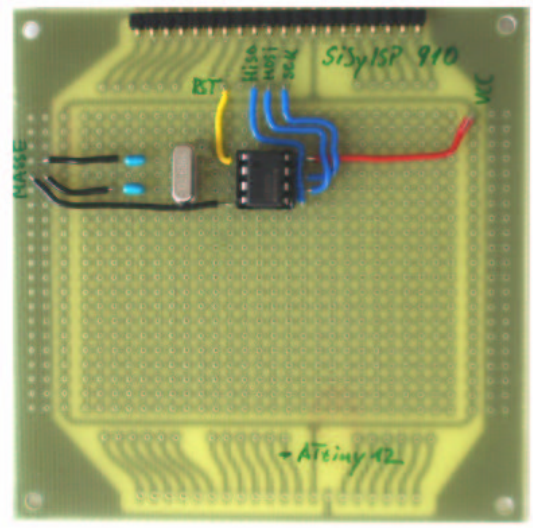


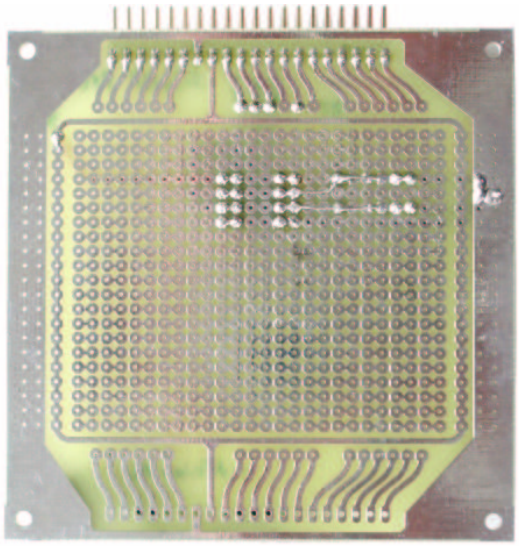
2. Verbinden Sie die PINs 5, 6 und 7 des IC Sockels mit Port B 3, 4 und 5 an der Steckerleiste (blau) und PIN 8 des IC Sockels mit der Versorgungsspannung (rot).





3. Ergänzen Sie Quarz und Kondensator und verbinden Sie PIN 1 des IC Sockels mit Port B.1 an der Steckerleiste (gelb) und PIN 2 und 3 des IC Sockels mit dem Quarz.





4. Stellen Sie alle Verbindungen entsprechend des Schaltplanes her. Danach kann der Controller (z.B.ATtiny13) in den IC Sockel eingesetzt werden.

4 myTinyProg Firmware

Die AVR910 Firmware für dieses Projekt ist als HEX-Datei *myTinyProg.hex* im Downloadbereich von www.myAVR.de verfügbar. Diese unterstützt unteranderem die Controller AT90S2323, ATtiny12, ATtiny13, ATtiny15 aber auch eine Reihe Mega AVR und setzt die oben beschriebene Schaltung voraus.

Port B.1 = RST
Port C.0 = rote LED (per Patchkabel auf dem myAVR-Board)
Port C.2 = grüne LED (per Patchkabel auf dem myAVR-Board)
Port B.3 = MISO
Port B.4 = MOSI
Port B.5 = SCK

4.1 Beschreibung der Firmware

Es werden unter anderem folgende AVR910 Kommandos unterstützt:

- Enter programming mode	'P'
- Set address	'A'
- Write program memory, low byte	'c'
- Write program memory, high byte	'C'
- Read program memory	'R'
- Write data memory	'D'
- Read data memory	'd'
- Chip erase	'e'
- Leave programming mode	'L'
- Read signature bytes	's'
- Return supported device codes	't'
- Return software identifier	'S'
- Return sofware version	'V'
- Return hardware version	'v'
- Return programmer type	'p'
- Set LED	'x'
- Clear LED	'y'

Der Betriebszustand des *myTinyProg* wird über zwei LEDs angezeigt

- rote LED = betriebsbereit,
- grüne LED = Brennvorgang.

Die Einstellungen für den COM-Port des PC´s müssen sein:

- 19200 Baud,
- 8 Datenbits,
- 1 Stoppbit,
- keine Parität sein.

4.2 Brennen der Firmware

Die Firmware muss zuerst wie bisher über den LPT-Port oder mySmartUSB auf den Mega8 des myAVR Boards gebrannt werden.

4.3 Brennen (Programmieren) des Tiny's

myTinyProg wird mittels der Buchsenleiste mit dem myAVR Board verbunden.

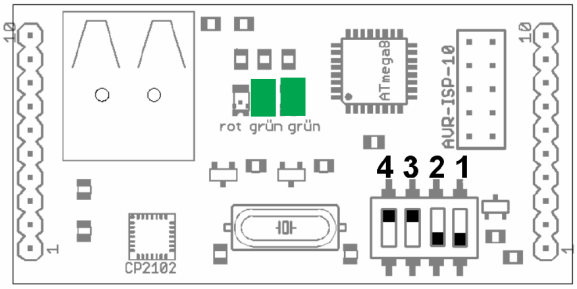
myAVR Board 1 LPT

Das LPT-Kabel ist abzuziehen und der serielle Anschluss wird zum Programmieren des Tiny's benutzt.

myAVR Board 2 USB

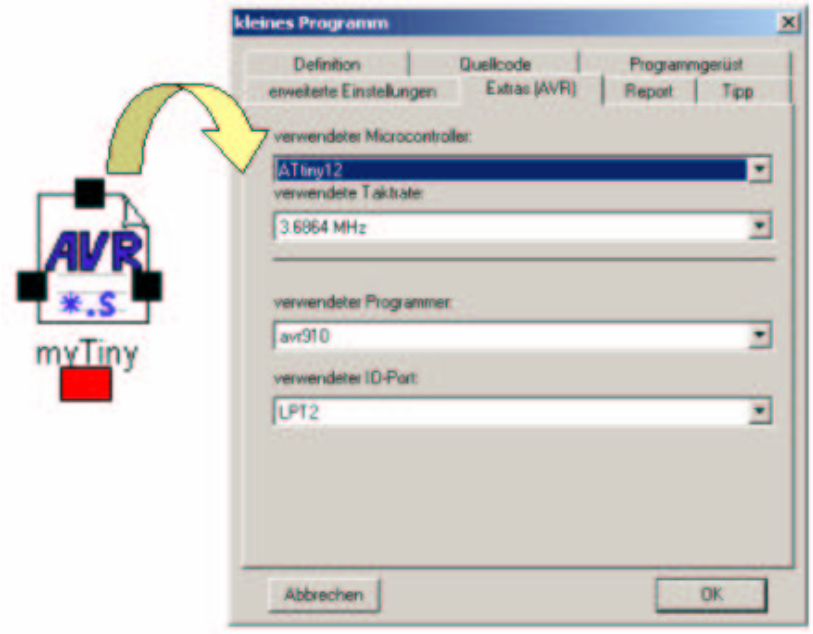
Zum Brennen über die USB-Schnittstelle ist der Datenmodus zu erzwingen (vgl. Abbildung). Das erfolgt mit dem Setzen der DIP-Schalter auf der USB-Platine mySmartUSB.

Datenmodus erzwingen
- DIP 2 ON
- DIP 3 OFF
- DIP 4 OFF



4.4 myTinyProg mit SiSy AVR

In SiSy AVR können Sie *myTinyProg* jetzt als AVR910 Programmer ansprechen. Dazu sind auf dem entsprechenden Programmmodul im Dialog „Definieren" unter „Extras (AVR)" die Eintragungen für den verwendeten Mikrocontroller, die Takt-rate, den Programmer und den IO-Port einzutragen. Die Abbildung zeigt ein mögliches Beispiel.



4.5 myTinyProg mit myAVR Workpad PLUS

Mit dem Programm myAVR Workpad PLUS können Sie jetzt ebenfalls myTinyProg als AVR910 Programmer ansprechen. Die Eintragungen für den verwendeten Mikrocontroller, Programmer und Port können Sie unter der Menüfolge *Extras/Einstellungen* vornehmen oder Sie hinterlegen diese Angaben im Dateikopf. Entsprechende Hinweise dazu erhalten Sie ebenfalls unter der Menüfolge *Extras/Einstellungen*.

5 Quellenverzeichnis

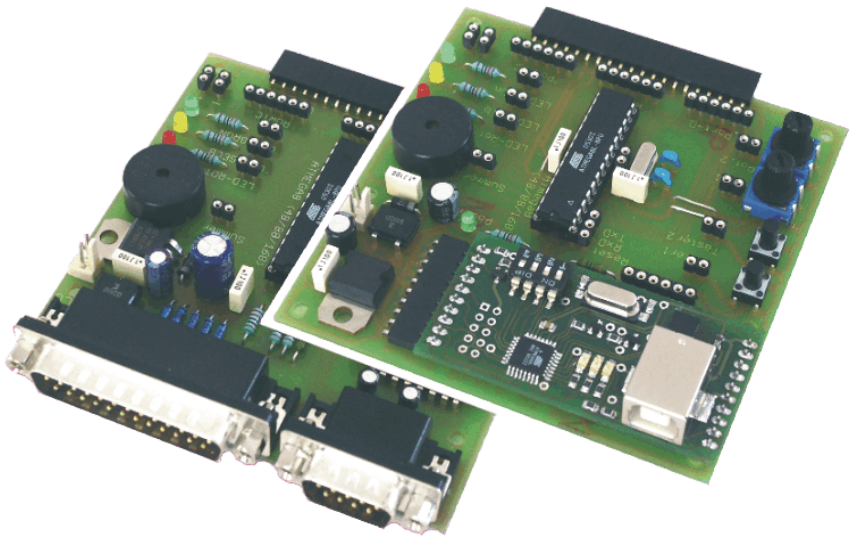
www.atmel.com
http://www.atmel.com/dyn/resources/prod_documents/DOC0931.PDF
http://www.atmel.com/dyn/resources/prod_documents/doc2521.pdf
http://www.atmel.com/dyn/resources/prod_documents/doc2585.pdf
http://www.atmel.com/dyn/resources/prod_documents/DOC0943.PDF
http://www.atmel.com/dyn/resources/prod_documents/AVR910.zip
http://www.atmel.com/dyn/resources/prod_documents/doc1006.pdf
http://www.atmel.com/dyn/resources/prod_documents/doc2535.pdf
http://www.atmel.com/dyn/resources/prod_documents/doc1187.pdf

www.sisy.de
www.myAVR.de

Dipl. Ing. Toralf Riedel
Dipl. Ing. Päd. Alexander Huwaldt

myAVR Lehrbuch Mikrocontroller-programmierung

Hardwarenahe
Programmierung von
AVR-Mikrocontrollern in
Assembler, C und BASCOM



Leseprobe

Inhalt

1 Einleitung 7
2 Vorbereitung 12
2.1 Hard- und Softwarevoraussetzungen 12
2.2 Beschaffen bzw. Herstellen der Hardware 12
2.3 Beschaffung und Installation der Software 15
2.3.1 Installation von SiSy AVR 15
2.3.2 Installation myAVR Workpad 15
3 Grundlagen zu Mikrocontrollern 16
3.1 Hardwaregrundlagen 16
3.1.1 Prozessorkern (Zentraleinheit) 17
3.1.2 Peripherie-Bausteine 18
3.1.3 Speicherarten und Speicherarchitektur 19
3.1.4 Prozessoren 20
3.1.5 Signale am AVR 21
3.2 Experimentierhardware myAVR Board 22
4 Programmierung von Mikrocontrollern 24
4.1 Programmiersprachen, Programmiergeräte, Schnittstellen 24
4.2 Mikrocontroller-Entwicklungsumgebung SiSy AVR 28
4.3 Schnellstart mit SiSy AVR 30
4.4 Entwicklungsumgebung myAVR Workpad 34
4.5 Schnellstart mit dem myAVR Workpad 36
5 Mikrorechnerprogrammierung in Assembler 40
5.1 Einführung in den AVR Assembler 40
5.2 Grundaufbau eines AVR-Assembler Programms 42
5.3 Der Befehlssatz des AVR-RISC-Assembler 47
5.4 Das seltsame „Hallo Welt“ eines Mikrocontrollers 49
5.5 Eingaben und Ausgaben 57
5.6 Eingabe, Verarbeitung, Ausgabe, EVA und der µC 64
5.7 Programmsteuerung und Algorithmen in Assembler 70
5.8 Unterprogramme 79
5.9 Interruptsteuerung 86
5.9.1 Polling oder Interrupt 86
5.9.2 Beispiel: „Lichtschalter mit Interrupt“ 92
5.9.3 Unsaubere Quellen prellen 96
5.9.4 Beispiel: „Lichtumschaltung mit Interrupt“ 97
5.10 Fequenzen und Timer 100
5.10.1 „Es ist Zeit für mehr Rhythmus“ 100
5.10.2 Beispiel: „Signalton“ 101
5.10.3 Es geht auch eleganter! 105
5.10.4 Beispiel: „Signalton mit Interrupt“ 108
5.11 Serielle Kommunikation, UART 113
5.11.1 UART initialisieren 115
5.11.2 UART, Daten empfangen 116
5.11.3 UART, Daten senden 117
5.11.4 Beispiel: „Ich sende !“ 117
5.12 Konstanten, Variablen, Speicher und EEPROM 119
5.12.1 Variablennamen für Register 120
5.12.2 Konstanten im Programmspeicher 121
5.12.3 Variablen und Daten im SRAM 124
5.12.4 Daten im EEPROM 127

5.13 Verarbeitung analoger Signale, Anna und der µC 131
5.13.1 Der Comparator 131
5.13.2 Der Analog/Digital-Wandler 137
5.13.3 Pulsweitenmodulation (PWM) 145
5.14 Watchdog-Timer, den Hund an die Leine legen 151
6 Kurzeinführung in die AVR-Programmierung mit C 155
6.1 Grundstruktur eines AVR-C Programms 155
6.2 Ausgaben in C 155
6.3 Variablen in C 156
6.4 Eingaben in C 156
6.5 Beispiel Ein- und Ausgabe in C 156
6.6 UART in C 157
6.7 Der Comparator in C 158
6.8 Interrupts in C 159
7 Einführung in die AVR-Programmierung mit BASCOM 161
7.1 Installation und Vorbereitung 161
7.2 Grundstruktur eines AVR BASCOM-Programms 165
7.3 BASCOM-Befehlssatz 165
7.4 Compilieren und Brennen mit BASCOM 166
7.5 Ausgaben in BASCOM 168
7.6 Variablen und deren Verarbeitung in BASCOM 169
7.7 Eingaben in BASCOM 170
7.8 Unterprogramme in BASCOM 171
7.9 UART Programmierung in BASCOM 173
7.10 Interrupts in BASCOM 175
7.11 Der Comparator in BASCOM 176
7.12 Der A/D-Wandler in BASCOM 177
8 Anhang zum Lehrbuch 179
8.1 Tabellen und Übersichten 179
8.2 Befehlssatz 182
8.3 Experimentierplattform myAVR Board 1.5 LPT 183
8.4 Experimentierplattform myAVR Board 2 USB 184
8.5 Quellen 185
8.6 Weitere Listings/Entwurfsmuster 187
9 Sachwortverzeichnis 198

Vorwort

Das vorliegende Lehrmaterial ist so konzipiert, dass Sie Schritt für Schritt in die hardwarenahe Programmierung eingeführt werden. Die aufgeführten Grundprinzipien, Problemstellungen, Fallstudien und Lösungsansätze gelten prinzipiell für eingebettete Prozessoren in kleinen und kleinsten Geräten und wenigen Ein- und Ausgabegeräten über Personalcomputern bis hin zu Großrechnern. Die Aufgaben die sie zu lösen haben, werden in „größeren“ Systemen durch das entsprechende Betriebsystem oder aufwändige Hardware erledigt. Bei Mikrocontrollerlösungen, in Embedded-Systems wie in Waschmaschinen, Autos, Telefonen, Steuerung von Anlagen und Robotern usw. finden Sie oftmals kein standardisiertes Betriebssystem. Dort müssen diese Aufgaben vom Entwickler selbst gelöst werden. Dafür sind diese eingebetteten Systeme hochspezialisiert, sowie bauteil- und energieeffizient bei geringstem Platzbedarf. Bedenkt man die weite Verbreitung von Mikrocontrollern, erkennt man schnell, dass hardwarenahe Programmierung keine Aufgabe von wenigen Betriebssystementwicklern, sondern ein weitverbreitetes Aufgabenspektrum in unterschiedlichsten Branchen und Berufsbildern, ist. Automatisierungstechnik, Mechatronik, Kommunikationstechnik, Medizintechnik, Automobilindustrie, Elektrotechnik, Maschinenbau usw.; kaum eine Branche kann heute noch auf die Anwendung von Mikrocontrollern verzichten. Der PC auf unserem Arbeitsplatz ist nur die berühmte Spitze des Eisberges bei der Anwendung von Prozessoren und der dazu notwendigen Software.
In diesem Lehrbuch wird die hardwarenahe Programmierung am Beispiel einer Mikrocontrollerlösung (Referenzhardware) erarbeitet. Das hat nicht nur den Grund, dass Mikrocontrollern eine zunehmende Bedeutung zukommt, sondern auch dass eine solche Lösung im Vergleich zu einem PC eine überschaubare und im Detail leicht verständliche Hardware besitzt.

Weitere Informationen und Beispiele finden Sie unter
www.myAVR.de.
Mit Ihren Fragen können Sie sich direkt an die SiSy-Hotline wenden,
hotline@myAVR.de.

Beim Lernen wünschen wir Ihnen viel Erfolg.

Ihr myAVR-Team

1 Einleitung

Mikrocontroller oder Mikrorechner?

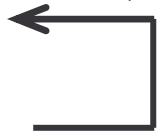
Ein Mikrocontroller ist ein Prozessor, bei dem im Unterschied zu PC-Prozessoren (Mikrorechnern) Speicher, wichtige Baugruppen wie Zeitgeber, digitale sowie analoge Ein- und Ausgabegeräte, auf einem einzigen Chip integriert sind, so dass eine Mikrocontroller-Lösung oft mit einigen wenigen externen Bauteilen auskommt. Ein PC-Prozessor verfügt nicht über eigene Eingabe- und Ausgabekomponenten, sondern über eine Verbindung zu einem externen Systembus, an dem die Ein- und Ausgabegeräte zum Beispiel als Steckkarten angeschlossen sind. In immer mehr Geräten des Alltags werden die Aufgaben von analogen Schaltungen durch Mikrocontroller realisiert. Damit lassen sich vor allem die Produktionskosten der Hardware drastisch senken. Prinzipiell kann die Struktur und Arbeitsweise von Mikrorechnern und Mikrocontrollern mit der von-Neumann-Architektur erklärt werden.

von-Neumann-Architektur (ausgewählte Aspekte):

- Ein Rechner besteht aus einer Zentraleinheit mit Rechenwerk und Steuerwerk, Eingabegeräten, Ausgabegeräten und einem Arbeitsspeicher;
- Die intern verwendete Signalmenge ist binär codiert (Digitalrechner);
- Der Rechner verarbeitet Worte fester Länge (Verarbeitungsbreite, Bussystem, Register, Speicher);
- Die Programmbefehle werden sequenziell in der Reihenfolge ihrer Speicherung abgearbeitet.

Befehlszyklus:

1. `POWER-ON` oder `RESET`
2. lade Befehlszeiger mit fester Startadresse (z.B.: 0x00)
3. lese Befehl (von Adresse im Befehlszeiger)
4. decodiere Befehl (Steuerwerk)
5. führe Befehl aus (Rechenwerk)
6. erhöhe den Befehlszeiger um 1, weiter mit 3.



- Der Rechner arbeitet taktgesteuert;
- Die sequenzielle Verarbeitung von Befehlen kann durch Sprunganweisung geändert werden;
- Befehlszeiger = Sprungadresse, Sprung mit oder ohne Bedingung, Unterprogrammaufruf mit Speicherung der Rücksprungadresse.

Prozessoren werden als erstes an der Verarbeitungsbreite der internen Architektur unterschieden: 8 Bit, 16 Bit, 32 Bit, 64 Bit usw. Diese Bit-Zahl kann man als die Länge der Datenworte interpretieren, die der Controller/Prozessor in einem Befehl verarbeiten kann. Die größte in 8 Bit (=1 Byte) darstellbare Zahl ist die 255. Somit kann ein 8-Bit-Mikrocontroller z.B. in einem einfachen Additionsbefehl nur Zahlen von 0 bis 255 verarbeiten und das Ergebnis kann ebenfalls nicht größer als 255 sein, sonst liegt ein Ergebnis-Überlauf vor. Zur Bearbeitung von größeren Zahlen werden dann mehrere Befehle hintereinander benötigt, was bei diesem Prozessor natürlich länger dauert.

Ein Prozessor benötigt eine meist extern eingespeiste Taktfrequenz. Dieses Taktsignal gibt dem Prozessor vor, wann der nächste Befehlsschritt ausgeführt werden soll. Da die Baugruppen innerhalb des Prozessors weder unendlich schnell noch gleich schnell arbeiten, ist dieses Synchronisationssignal für die Zusammenarbeit von Steuerwerk, Rechenwerk, Ein- und Ausgabegeräten notwendig. Die maximale Frequenz, mit der ein Prozessor betrieben werden kann, reicht heute von 1 MHz bis hin zu über 3 GHz.

Bei Mikrocontrollern sind im Moment Taktfrequenzen von 1 bis 300 MHz üblich (Stand: Februar 2004). Diese Taktfrequenz sagt jedoch noch nichts über die tatsächliche Geschwindigkeit eines Prozessors aus. Es ist auch entscheidend, ob für den oben beschriebenen Befehlszyklus viele, wenig oder gar nur ein Taktzyklus benötigt wird.
Bei vielen Prozessoren wird die Frequenz intern geteilt, ein mit 24 MHz getakteter Mikrocontroller der Intel Reihe 8051 arbeitet eigentlich nur mit 2 MHz, da der externe Takt durch 12 geteilt wird. Benötigt dieser dann für einen Befehl durchschnittlich 2 Taktzyklen, so bleiben "nur" noch 1 Mio. Befehle pro Sekunde übrig (1 MIPS, Million Instructions per Second). Dieser Sachverhalt trifft besonders auf Prozessoren zu, die eine CISC-Architektur besitzen (Complex Instruction Set Computer). Betrachten wir dazu im Gegensatz eine RISC-Architektur (Reduced Instruction Set Computer), finden wir hier kaum interne Taktteilungen und für die Ausführung eines Befehls wird oft nur ein Taktzyklus benötigt. Ein moderner RISC-Mikrocontroller, der ungeteilt mit 8 MHz arbeitet und für viele Befehle nur einen Prozessorzyklus braucht, schafft gegenüber dem mit 24 MHz getakteten CISC Controller fast 8 Mio. Befehle pro Sekunde (8 MIPS). Ein RISC-Prozessor hat dafür einen geringeren Befehlssatz. Komplexe Befehle wie eine Division, müssen deshalb algorithmisch umgesetzt werden.

Wozu ist ein Mikrocontroller gut?

Welche Aufgaben Mikrocontroller haben und warum sie eine zunehmende Bedeutung erlangen, soll folgender Einsatzfall verdeutlichen. Uns allen sind Detektoren unterschiedlichster Art bekannt. Am auffälligsten sind die Portalmetalldetektoren in Flughäfen oder wichtigen öffentlichen Gebäuden, aber auch der stationäre Blitzer um die Ecke verfügt als Sensor über einen Metalldetektor in der Fahrbahn. Ein anderer Anwendungsfall ist zum Beispiel die Lebensmittelindustrie. Es ist sehr wichtig für den Hersteller von Backmischungen, dass keine Nägel, Schrauben, Metallsplitter oder ähnliches während des Produktionsprozesses in das Nahrungsmittel gelangt. Auf der Verpackungsstrecke werden alle Packungen, in denen der Metalldetektor Metallteile ortet, automatisch ausgesondert. Metalldetektoren werden in großen Stückzahlen für unterschiedlichste Anwendungsfälle benötigt.
Das folgende Schema zeigt die Funktionseinheiten eines Metalldetektors nach dem Pulsinduktionsverfahren (PI).

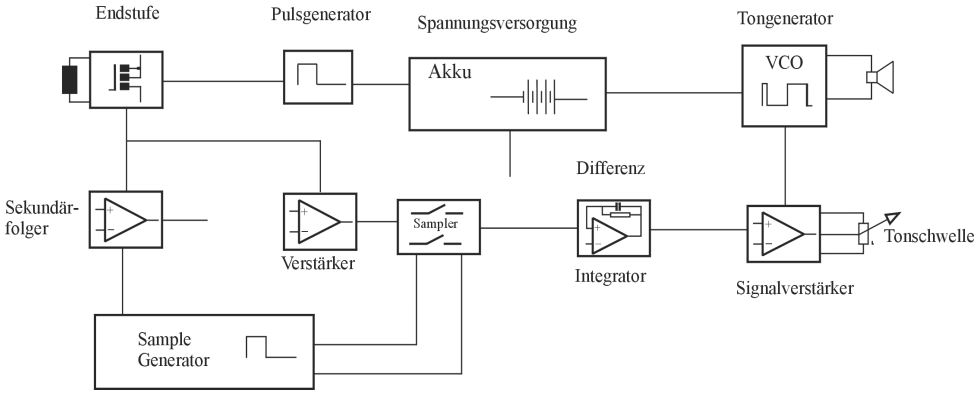


Abbildung: Funktionsschema eines PI-Metalldetektors

Der klassische Aufbau eines solchen Detektors als diskrete analoge Schaltung erfordert trotz Verwendung integrierter Schaltungen einen bestimmten, kaum weiter reduzierbaren Materialaufwand an aktiven und passiven Bauelementen.
Die Materialkosten für diesen konkreten Metalldetektor belaufen sich auf ca. 25 € (Stand: Februar 2004).

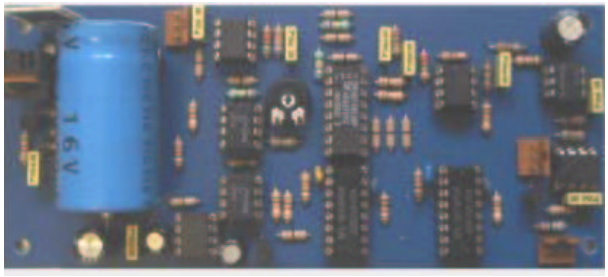


Abbildung: vollständig analoger PI-Metalldetektor

Einen nicht unerheblichen Teil dieser diskreten Schaltung macht die Realisierung des „Timings“, also der Steuerung der verschiedenen Komponenten für den Detektor aus. Der folgende Detektor hat das gleiche Arbeitsprinzip und die gleiche Leistungsfähigkeit wie der oben gezeigte Detektor. Der Materialaufwand ist jedoch um ein vielfaches geringer (Materialkosten ca. 8 €). Dies wurde erreicht, indem die Steuerung des Detektors von einem Mikrocontroller und der entsprechenden Software darauf übernommen wurde.

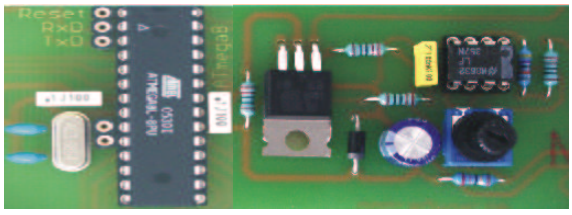


Abbildung: PI-Metalldetektor mit Mikrocontroller

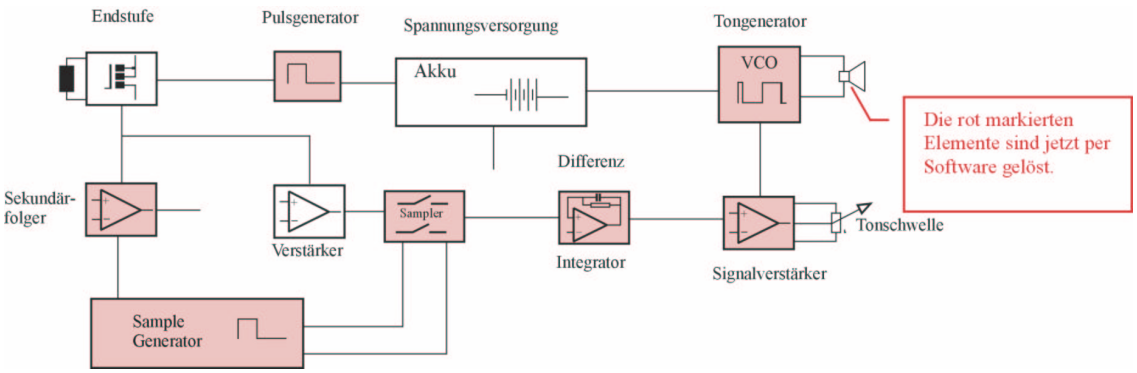


Abbildung: Funktionsverlagerung von der Hardware in die Software

Die zuvor diskret mit analogen Bauelementen realisierten Funktionen des Detektors werden in der mikrocontrollerbasierenden Lösung in die Software verlagert. Damit reduziert sich die Anzahl der aktiven und passiven analogen Bauelemente. Die Komplexität und Größe der Leiterplatte konnte reduziert werden. In der Gesamtheit konnten die Kosten für die Herstellung eines Detektors auf ein Drittel reduziert werden. Die wirtschaftliche Bedeutung der Reduzierung der Stückkosten, in diesem Fall um 60%, braucht man nicht weiter zu betonen. Aber selbst wenn die Kostenreduzierung durch den Mikrocontrollereinsatz pro System nur einige

Cent beträgt, liegt der Kostenreduzierungseffekt in der Massenfertigung schnell bei Millionen Euro.
Ganz klar, die Softwarelösung für den Tongenerator lässt sich auf jedes einzelne System kopieren und verursacht keine Beschaffungskosten mehr. Die diskrete Lösung benötigt eine integrierte Schaltung (IC), 3 Widerstände und einen Kondensator mehr; und diese müssen für jedes zu produzierende Gerät neu beschafft werden. Und das ist noch nicht alles. Die Mikrocontrollerlösung könnte sogar beliebige Melodien abspielen, die der Hersteller im Internet zum herunterladen anbieten kann. Solche zusätzlichen Eigenschaften werden von den Anwendern der Lösungen gern angenommen wie der aktuelle Trend bei Klingeltönen und Handylogos zeigt. Es werden nicht nur Kosten, Platz und Gewicht gespart, sondern zusätzliche Möglichkeiten eröffnet. Daher auch der ungebrochene Trend, selbst einfache Geräte wie eine Kaffeemaschine mit einem Mikrocontroller auszustatten.
Hier ein paar Beispiele, für welche Aufgaben Mikrocontroller verwendet werden können: Roboter, CD-, MP3- und DVD-Player, Temperaturregler, Füllstandsregler, Motorsteuerungen, Signaldecoder für Satellitenempfang, Fernbedienung, Alarmanlagen, Schaltuhren, Ladegeräte, Waschmaschinen, Geschirrspüler, Fernseher, Radio, Wecker/Uhr, Messwerterfassung (z.B. Drehzahlmessung im Auto), intelligente Geräte in der Automatisierungstechnik, intelligente Sensoren, intelligente Aktoren z.B. die Airbags im PKW, Handy, alle Formen von Heimelektronik, Geräte der Medizintechnik, Spielzeug …

Abbildung: Internetfähiger TOASTER ;-)

Anforderungen und Möglichkeiten von Mikrocontrollerlösungen
- programmierbar (Update, Optimierung, Wartung)
- flexible Schnittstellen (vielfältig, integriert, standardisiert)
- Selbstdiagnose, Fehlerkorrektur, Debuginterface
- Echtzeitfähigkeit (schnelle Reaktionszeiten)
- Timer, Interruptfähigkeit
- deterministisch (bestimmbares, berechenbares Verhalten)
- geringe Kosten, geringer Leistungsverbrauch

Die Anwendungsgebiete von Mikrocontrollern sind schier unendlich. In allen Bereichen unseres Lebens lassen sich heute „versteckte“ Mikrocontroller finden.

Welchen Mikrocontroller verwenden?

Mikrocontroller werden von unterschiedlichen Firmen angeboten. Weit verbreitete Mikrocontroller sind die der Intel-Reihe 8051, der Zilog-Reihe Z80, Z86, die PIC-Controller der Firma Microsystems und die AVR-Controller der Firma Atmel. Die AVR-Reihen von Atmel haben eine innovative RISC-Architektur, die schnell und einfach zu erlernen ist. Sie sind inzwischen sehr weit verbreitet. Sie sind elektrisch robust und bis zu 10.000-mal programmierbar. Da die AVR-Prozessoren zu den modernsten Controllern am Markt gehören und enorme Zuwachsraten aufweisen, sollen sich alle Ausführungen und die Experimentierhardware auf diese Controller beziehen. Prinzipiell lassen sich jedoch alle Aussagen auf alle anderen Mikrocontroller übertragen.
Ein pikanter Hintergrund ist, dass der AVR-Kern eine Entwicklung von zwei Studenten der Universität Trontheim in Norwegen ist. Atmel kaufte die Lizenz und

entwickelte dieses innovative Konzept weiter. Hartnäckig hält sich das Gerücht, dass die Abkürzung „AVR“ etwas mit den Vornamen der beiden inzwischen nicht mehr Studenten **A**lf Egil Bogen und **V**egard Wollan zu tun hat, die diesen **R**ISC-Prozessor entwickelt haben. Wer weiß ;-)

Welche Programmiersprache benutzen? Assembler, C oder BASIC?

Assembler ist für den Einstieg in die Programmierung von Mikrocontrollern zwar recht schwierig, aber am besten geeignet. Da Assemblerprogramme auf Maschinencodeebene angesiedelt sind, lernt man den Aufbau und Funktionsweise eines Prozessors richtig kennen und kann ihn dadurch optimal ausnutzen. Zudem stößt man bei jedem Compiler irgendwann einmal auf Problemstellungen, die sich nur durch das Verwenden von Assemblercode lösen lassen. Die zunehmende Leistungsfähigkeit und immer mehr Speicher auf den Mikrocontrollern erlauben inzwischen auch die Anwendung von höheren Programmiersprachen. Die Sprachen C/C++ bieten sich auf Grund ihrer Effizienz und Hardwarenähe für die Programmierung von komplexen Mikrocontrollerlösungen an.

Der AVR Assembler wird Ihnen Schritt für Schritt und verständlich vermittelt. Mit den so erworbenen Kenntnissen ist die Programmierung von AVR-Controllern mit jeder anderen Sprache noch effizienter. Die Auseinandersetzung mit den Maschinenbefehlen und der Programmierung in Assembler führt zum besseren Verständnis des Aufbaus und der Arbeitsweise des Prozessors/Mikrocontrollers.

```
;-----------------------------------------------------------------
;* Titel             :Programmvorlage für myAVR Board
;* Prozessor         :ATmega8 mit 3,6864 MHz
;* Datum             :15.03.2004
;* Autor             :Dipl. Ing. Päd. Alexander Huwaldt
;-----------------------------------------------------------------
.include            "avr.h"
;-----------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:       rjmp    main        ; 1    POWER ON RESET
             reti                ; 2    Int0-Interrupt
             reti                ; 3    Int1-Interrupt
             reti                ; 4    TC2 Compare Match
             reti                ; 5    TC2 Overflow
             reti                ; 6    TC1 Capture
             reti                ; 7    TC1 Compare Match A
             reti                ; 8    TC1 Compare Match B
             reti                ; 9    TC1 Overflow
             reti                ; 10   TC0 Overflow
             reti                ; 11   SPI, STC Serial Transfer Complete
             reti                ; 12   UART Rx Complete
             reti                ; 13   UART Data Register Empty
             reti                ; 14   UART Tx Complete
             reti                ; 15   ADC Conversion Complete
             reti                ; 16   EEPROM Ready
             reti                ; 17   Analog Comparator
             reti                ; 18   TWI (I²C) Serial Interface
             reti                ; 19   Store Program Memory Ready
;-----------------------------------------------------------------
; Start, Power ON, Reset
main:        ldi    r16,lo8(RAMEND)
             out    SPL,r16             ; Init Stackpointer LO
             ldi    r16,hi8(RAMEND)
             out    SPH,r16             ; Init Stackpointer HI
             ; Hier den Init-Code eintragen
;-----------------------------------------------------------------
mainloop:    wdr
             ; Hier den Quellcode eintragen
             rjmp   mainloop
;-----------------------------------------------------------------
```

Abbildung: AVR Assemblerprogramm

...

4 Programmierung von Mikrocontrollern

4.1 Programmiersprachen, Programmiergeräte, Schnittstellen

Für die Programmierung von Mikrocontrollern stehen unterschiedliche Entwicklungsumgebungen für verschiedene Sprachen zur Verfügung. Diese weisen eine Reihe gemeinsamer Merkmale auf. Der wichtigste Aspekt, der hier angesprochen werden muss ist, dass im Gegensatz zur Programmierung eines PC-Programms das fertige Programm nicht auf der Entwicklungsplattform, also unserem PC-Arbeitsplatz gestartet, ausgeführt und getestet werden kann. Für das Ausführen der entwickelten Controllersoftware ist es notwendig, das ausführbare Programm in geeigneter Weise auf den Controller und diesen in das Zielsystem zu bringen. Neben dem üblichen Übersetzungslauf des Quellcodes ist ein weiterer Arbeitsschritt notwendig. Dabei wird das übersetzte Controllerprogramm in den Programmspeicher (FLASH) des Mikrocontrollers übertragen. Dieser Vorgang wird „Programmieren“ oder „Brennen“ genannt und erfordert eine spezielle Schnittstelle für das Programmieren des Controllers. Diese zusätzliche Hardware wird „Programmer“ genannt.
Eine Ausnahme bilden Simulatoren der Zielhardware. Bei diesen ist es möglich, ohne den Brennvorgang, das fertige Programm auf den PC zu testen.

Die am weitesten verbreiteten Sprachen für die Programmierung von AVR-Mikrocontrollern sind:

- Assembler
- C/C++
- BASIC (BASCOM)

Eine Entwicklungsumgebung enthält typischerweise folgende Komponenten:

- Quellcode-Editor
- Übersetzungsprogramm (Assembler, Compiler)
- Programmier- /Brenn-Programm
- Testwerkzeuge (Terminal, Simulator, Debugger)

Folgende Dateitypen finden Verwendung bei der Programmierung von Mikrocontrollern:

- *.ASM; *.S; *.C; *.CC; *.BAS
  Quellcode, wird im Editor erstellt und übersetzt (Compiler, Assembler) zu …
- *.O; *.OBJ
  Objektdateien, entstehen beim Übersetzen, werden zu ausführbaren Dateien gebunden (Linker) …
- *.HEX; *.BIN
  Binärdatein, entstehen beim Linken, werden beim Brennen in den Programmspeicher des Controllers übertragen.

Die folgenden Abbildungen zeigen jeweils die allgemeine Grundstruktur eines Mikrocontrollerprogramms in unterschiedlichen Programmiersprachen.

```
;-------------------------------------------------------------
;* Titel             :Programmvorlage für myAVR Board
;* Prozessor         :ATmega8 mit 3,6864 MHz
;-------------------------------------------------------------
.include            "avr.h"
;--- Reset and Interruptvectoren ; VNr. Beschreibung---
begin:       rjmp    main        ; 1    POWER ON RESET
             reti                ; 2    Int0-Interrupt
             reti                ; 3    Int1-Interrupt
             reti                ; 4    TC2 Compare Match
             reti                ; 5    TC2 Overflow
             reti                ; 6    TC1 Capture
             reti                ; 7    TC1 Compare Match A
             reti                ; 8    TC1 Compare Match B
             reti                ; 9    TC1 Overflow
             reti                ; 10   TC0 Overflow
             reti                ; 11   SPI, STC Serial Transfer Complete
             reti                ; 12   UART Rx Complete
             reti                ; 13   UART Data Register Empty
             reti                ; 14   UART Tx Complete
             reti                ; 15   ADC Conversion Complete
             reti                ; 16   EEPROM Ready
             reti                ; 17   Analog Comparator
             reti                ; 18   TWI (I²C) Serial Interface
             reti                ; 19   Store Program Memory Ready
;-------------------------------------------------------------------
; Start, Power ON, Reset
main:        ldi          r16,lo8(RAMEND)
             out          SPL,r16
             ldi          r16,hi8(RAMEND)
             out          SPH,r16            ; Init Stackpointer
             ; Hier den Init-Code eintragen
;-------------------------------------------------------------------
mainloop:    wdr
             ; Hier den Quellcode eintragen
             rjmp mainloop
;-------------------------------------------------------------------
```

Abbildung: AVR Assemblerprogramm

```
//-------------------------------------------------------------
// Titel        :Programmvorlage für das myAVR Board
// Prozessor    :ATmega8 mit 3,6864 MHz
//-------------------------------------------------------------
#include <io.h>
int main ()
{
   //hier Init-Code eingeben
   while (1) // Mainloop
   {
      //hier Steuer-Code eingeben
   }
return 0;
}
//-------------------------------------------------------------
```

Abbildung: AVR C-Programm

```
'-----------------------------------------------------------------------
' SiSy AVR.bas Vorlage für myAVR Board
' Grundstruktur eines µC-Programms
'-----------------------------------------------------------------------
$regfile = "m8def.dat"
$crystal = 3686400
' hier Initalisierung durchführen
Do
   ' hier Verarbeitungsaufgabe durchführen
Loop
End
```

Abbildung: AVR BASIC-Programm (BASCOM)

4.2 Mikrocontroller-Entwicklungsumgebung SiSy AVR

Diesen und den folgenden Abschnitt können Sie überspringen wenn Sie nicht SiSy AVR als Entwicklungsumgebung verwenden.

Schauen wir uns als nächstes kurz in der Entwicklungsumgebung SiSy AVR um. SiSy AVR ist ein allgemeines Entwicklungswerkzeug, mit dem man von der Konzeption eines Systems, bis zur Realisierung die verschiedensten Arbeitsschritte unterstützen kann. Für die Entwicklung von Mikrocontrollerlösungen bietet sich die einfache Programmierung an.

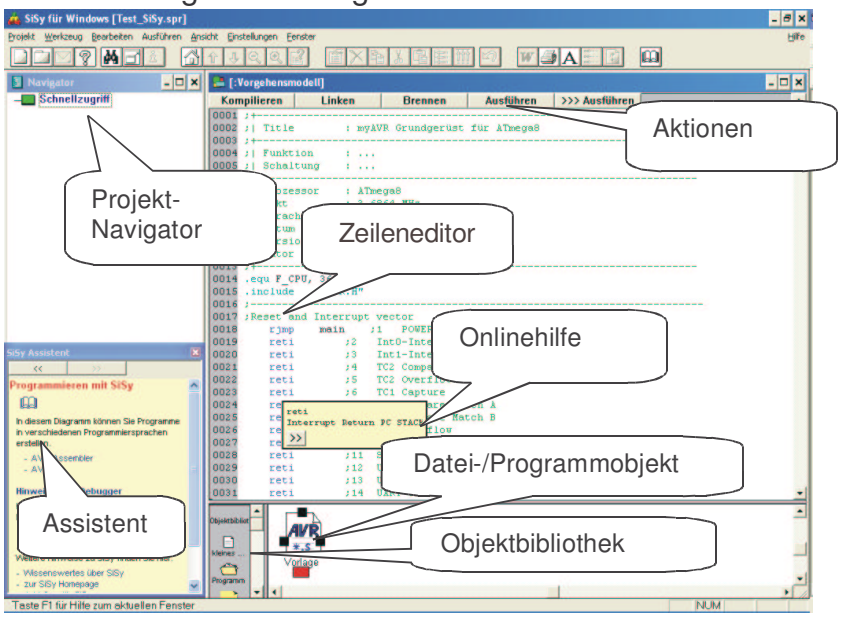


Abbildung: Bildschirmaufbau der Entwicklungsumgebung SiSy AVR

Beim Kompilieren, Linken und Brennen des Schnellstart-Beispiels öffnete sich ein Ausgabefenster und zeigte Protokollausgaben der Aktionen an. Wenn die Hardware ordnungsgemäß angeschlossen, von der Software erkannt und das Programm erfolgreich auf den Programmspeicher des Mikrocontrollers übertragen wurde, muss die letzte Ausschrift folgenden Inhalt haben:

```
avrdude: xxx bytes of flash written
…
avrdude: verifying …
avrdude: xxx bytes of flash verified
avrdude done: Thank you
Ende.
```

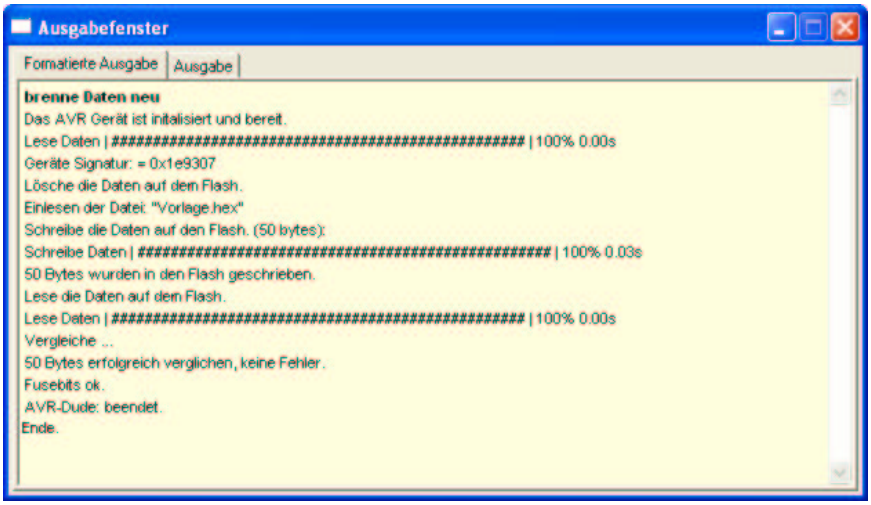

Ausgabefenster
Formatierte Ausgabe
Ausgabe
brenne Daten neu
Das AVR Gerät ist initialisiert und bereit.
Lese Daten | ########## | 100% 0.00s
Geräte Signatur: = 0x1e9307
Lösche die Daten auf dem Flash.
Einlesen der Datei: "Vorlage.hex"
Schreibe die Daten auf den Flash. (50 bytes):
Schreibe Daten | ########## | 100% 0.03s
50 Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Lese Daten | ########## | 100% 0.00s
Vergleiche ...
50 Bytes erfolgreich verglichen, keine Fehler.
Fusebits ok.
AVR-Dude: beendet.
Ende.

Abbildung: Ausgabefenster mit „Brenn“ - Protokoll

Die Inbetriebnahme, Test und Datenkommunikation mit der Mikrocontrollerlösung erfolgen über das myAVR-Controlcenter. Dabei wird über die Schaltfläche „Start“ das Testboard mit der nötigen Betriebsspannung versorgt und der Controller gestartet. Der Datenaustausch mit dem myAVR Board ist möglich, wenn das Null-Modemkabel (oder USB-Kabel) an Rechner und Testboard angeschlossen ist, sowie die Mikrocontrollerlösung dafür vorgesehen ist. Es können Texte und Bytes (vorzeichenlose ganzzahlige Werte bis 255) an das Board gesendet und Text empfangen werden. Die empfangenen Daten werden im Protokollfenster angezeigt.

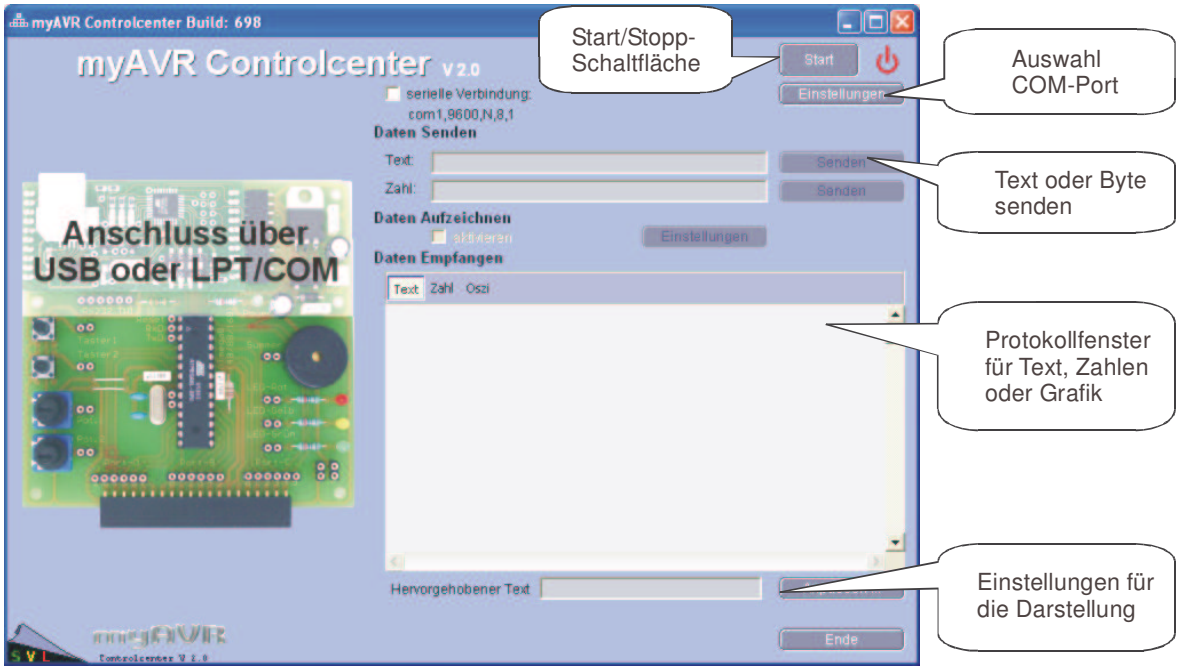


Abbildung: myAVR-Controlcenter

Nutzen Sie die zahlreichen Hilfen und Vorlagen, die SiSy AVR bietet!

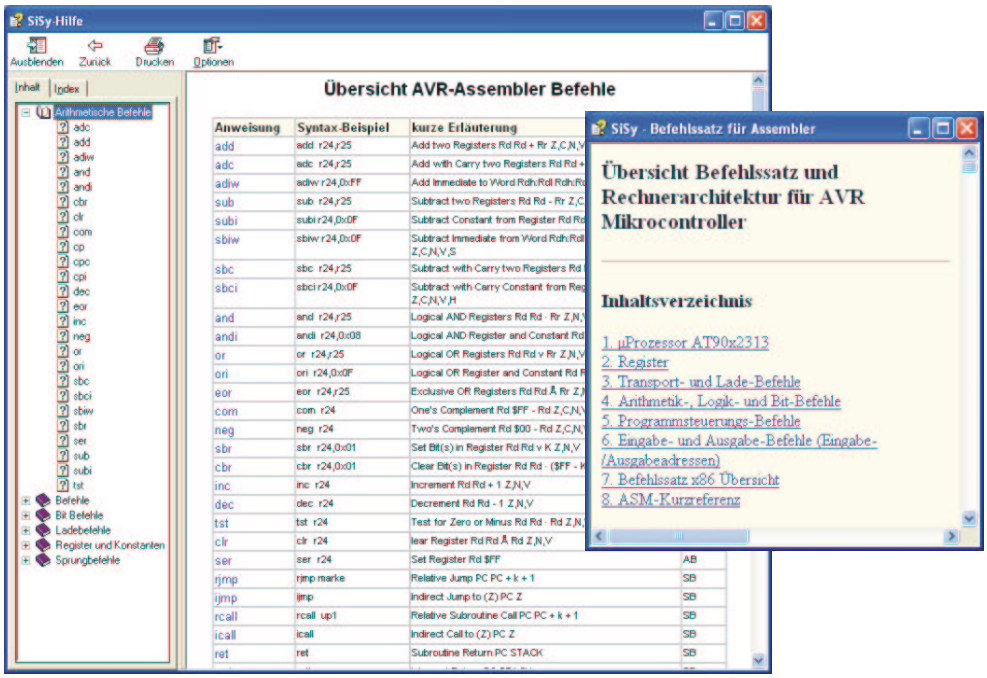


Abbildung: SiSy AVR-Assembler --> Dictionary und Hilfe

4.3 Schnellstart mit SiSy AVR

Voraussetzungen:

Für die Bearbeitung der folgenden Aufgaben benötigen Sie die aufgeführte Software und Hardware.

Software:

- SiSy AVR ab Version 2.17
- SiSy-Ausgabe AVR; Developer, Professional oder BS und das installierte *Add-On AVR*

Hardware:

- Ein bestücktes myAVR Board
- LPT-Verlängerungskabel oder USB-Kabel A-B
- 9 V Batterie / 9 V Netzteil bei Bedarf (z.B.: autonomer Einsatz)
- evtl. Nullmodemkabel für die COM Schnittstelle

Zielstellung:

Der Schnelleinstieg zur Mikrocontroller-Programmierung soll Ihnen helfen, das Add-On AVR kennen zu lernen und erste Schritte in der hardwarenahen Programmierung mit SiSy AVR zu gehen. Die Funktion des nachfolgenden Programms ist es, bei Programmstart alle optischen Ausgabegeräte (LED) auf dem myAVR Board zum leuchten zu bringen.

Ein neues Projekt anlegen

Starten Sie SiSy AVR und wählen Sie *Assistent öffnen*. Wählen Sie den Menüpunkt *Neues Projekt anlegen*, vergeben Sie den Projektnamen „Alle_Lichter_an“ und bestätigen Sie mit *OK*. Wählen Sie das Vorgehensmodell „Programmierung“ und laden Sie keine Diagrammvorlage.

Kleines Assembler-Programm anlegen

Erstellen Sie ein Programm für den AVR Mikrocontroller, in dem Sie per Drag & Drop aus der Objektbibliothek ein „kleines Programm“ in das Diagrammfenster ziehen. In dem aufgeblendeten Dialogfenster vergeben Sie den Namen „Alle_Lichter_an“. Der Datei- und Programmname wird dabei automatisch vergeben. Wählen Sie die Sprache „AVR Assembler“.

Über die Registerkarte „Programmgerüst“ können Sie die Vorlage „Grundgerüst“ für ein AVR Assemblerprogramm laden.

Abbildung: Dialog „kleines Programm“

```
;-------------------------------------------------------------------
;* Titel            :Programmvorlage für myAVR Board
;* Prozessor        :ATmega8 mit 3,6864 MHz
;* Schaltung        :...
;* Datum            :15.03.2004
;* Autor            :Dipl. Ing. Päd. Alexander Huwaldt
;-------------------------------------------------------------------
.include     "avr.h"
;-------------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:       rjmp    main       ; 1    POWER ON RESET
             reti               ; 2    Int0-Interrupt
             reti               ; 3    Int1-Interrupt
             reti               ; 4    TC2 Compare Match
             reti               ; 5    TC2 Overflow
             reti               ; 6    TC1 Capture
             reti               ; 7    TC1 Compare Match A
             reti               ; 8    TC1 Compare Match B
             reti               ; 9    TC1 Overflow
             reti               ; 10   TC0 Overflow
             reti               ; 11   SPI, STC Serial Transfer Complete
             reti               ; 12   UART Rx Complete
             reti               ; 13   UART Data Register Empty
             reti               ; 14   UART Tx Complete
             reti               ; 15   ADC Conversion Complete
             reti               ; 16   EEPROM Ready
             reti               ; 17   Analog Comparator
             reti               ; 18   TWI (I²C) Serial Interface
             reti               ; 19   Store Program Memory Ready
;-------------------------------------------------------------------
; Start, Power ON, Reset
main:        ldi    r16 , lo8(RAMEND)
             out    SPL , r16                ; Init Stackpointer LO
             ldi    r16 , hi8(RAMEND)
             out    SPH , r16
             ; Hier den Init-Code eintragen
;-------------------------------------------------------------------
mainloop:    wdr
             ; Hier den Quellcode eintragen
             rjmp mainloop
;-------------------------------------------------------------------
```

Abbildung: Grundgerüst AVR-Assembler-Programm

Quellcode in Assembler erstellen

Die Ausgabegeräte LED sollen vom Prozessorport D gesteuert werden. Die Realisierung erfolgt über Bits im Register R16.
Geben Sie folgenden Quellcode ein bzw. ergänzen Sie die Programmvorlage.

```
;+-----------------------------------------------------+
;| Titel         : Übung: „alle Lichter an“            |
;+-----------------------------------------------------+
;| Prozessor     : ATmega8  3,6864 MHz                 |
;| Sprache       : Assembler                           |
;| Datum         : 28.02.2006                          |
;| Autor         : Dipl. Ing. Päd. Alexander Huwaldt   |
;+-----------------------------------------------------+
.include    "avr.h"
;-------------------------------------------------------
;Reset and Interrupt vector   ;VNr.  Beschreibung
   rjmp    main              ;1     POWER ON RESET
   reti                      ;2     Int0-Interrupt
   reti                      ;3     Int1-Interrupt
   reti                      ;4     TC2 Compare Match
   reti                      ;5     TC2 Overflow
    reti                     ;6     TC1 Capture
   reti                      ;7     TC1 Compare Match A
   reti                      ;8     TC1 Compare Match B
   reti                      ;9     TC1 Overflow
   reti                      ;10    TC0 Overflow
   reti                      ;11    SPI, STC Serial Transfer Complete
```

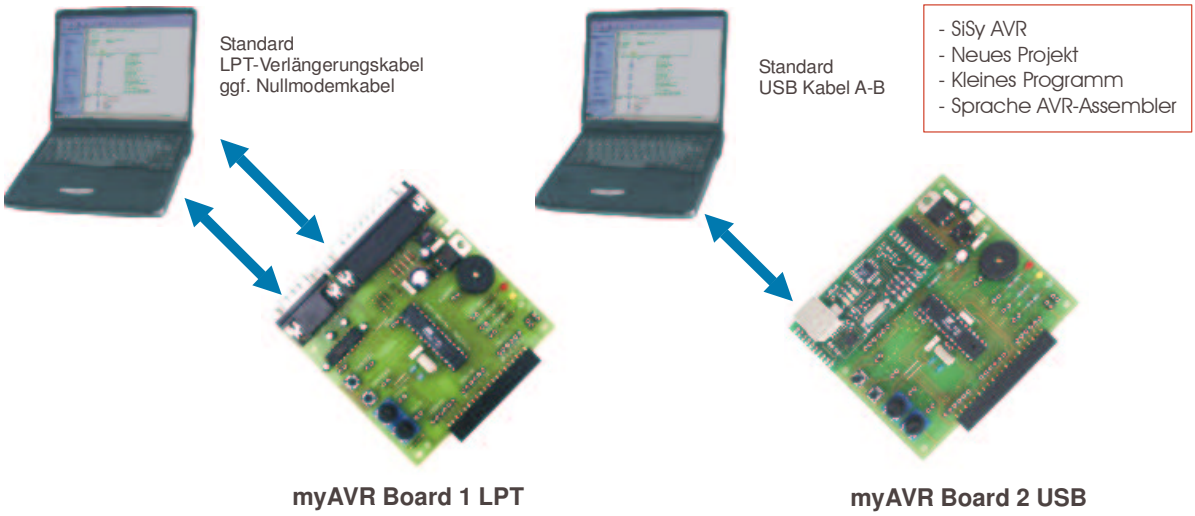
```
   reti                          ;12    UART Rx Complete
   reti                          ;13    UART Data Register Empty
   reti                          ;14    UART Tx Complete
   reti                          ;15    ADC Conversion Complete
   reti                          ;16    EEPROM Ready
   reti                          ;17    Analog Comparator
   reti                          ;18    TWI (IC) Serial Interface
   reti                          ;19    Store Program Memory Ready
;-----------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     r16,0b00011100      ; Bit 2-4 auf HIGH
            out     DDRD,r16            ; Port D.2-4 auf OUT
;-----------------------------------------------------------
mainloop:   wdr
            ldi     r16, 0b00011100     ;Bit 2-4 auf HIGH
            out     PORTD,r16           ;Port D.2-4 auf HIGH
            rjmp    mainloop
;-----------------------------------------------------------
```

Compilieren und Linken

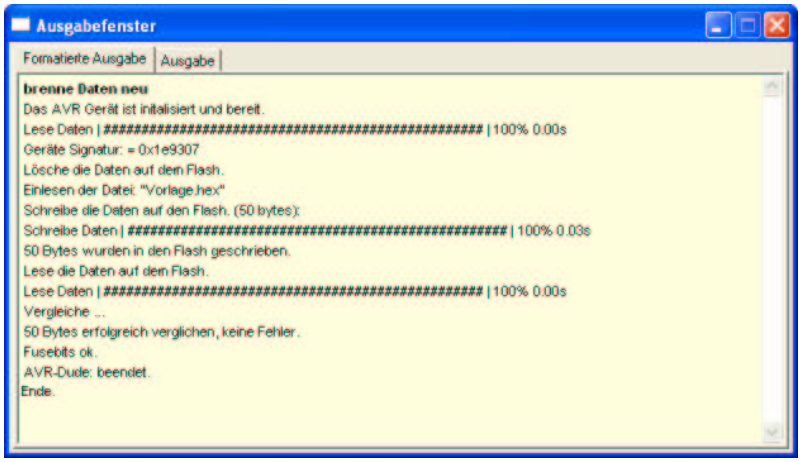
Der eingegebene Quellcode muss nun in Maschinencode für den AVR-Prozessor übersetzt werden. Wählen Sie dazu die Schaltflächen „Compilieren“ und „Linken“. Bei fehlerfreier Übersetzung liegt das Programm als „Alle_Lichter_An.hex“ vor und kann auf den FLASH-Programmspeicher des Prozessors gebrannt werden.

Hardware anschließen und brennen

Das myAVR Board verfügt über eine ISP (In System Programming) Schnittstelle. Der Prozessor muss also nicht für die Programmierung aus dem System entfernt werden, um ihn in einem gesonderten Programmiergerät zu brennen, sondern kann im myAVR Board direkt programmiert werden. Dazu schließen Sie das Programmierkabel an den LPT- oder USB-Port Ihres Rechners an.

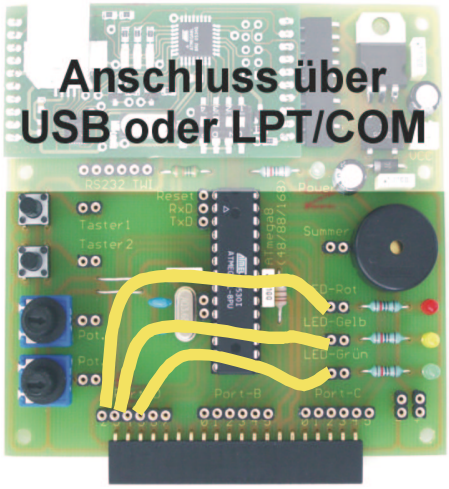


Zum Brennen wählen Sie die Schaltfläche „Brennen“. Bei erfolgreichem Brennvorgang erhalten Sie im Ausgabefenster folgende Meldung:


Ausgabefenster
Formatierte Ausgabe
Ausgabe
brenne Daten neu
Das AVR Gerät ist initialisiert und bereit.
Lese Daten | ## | 100% 0.00s
Geräte Signatur: = 0x1e9307
Lösche die Daten auf dem Flash.
Einlesen der Datei: "Vorlage.hex"
Schreibe die Daten auf den Flash. (50 bytes):
Schreibe Daten | ## | 100% 0.03s
50 Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Lese Daten | ## | 100% 0.00s
Vergleiche ...
50 Bytes erfolgreich verglichen, keine Fehler.
Fusebits ok.
AVR-Dude: beendet.
Ende.

Mikrocontrollerlösung testen

Um das Programm zu testen ist es nötig, den Port D mit den Ausgabegeräten LED zu verbinden.

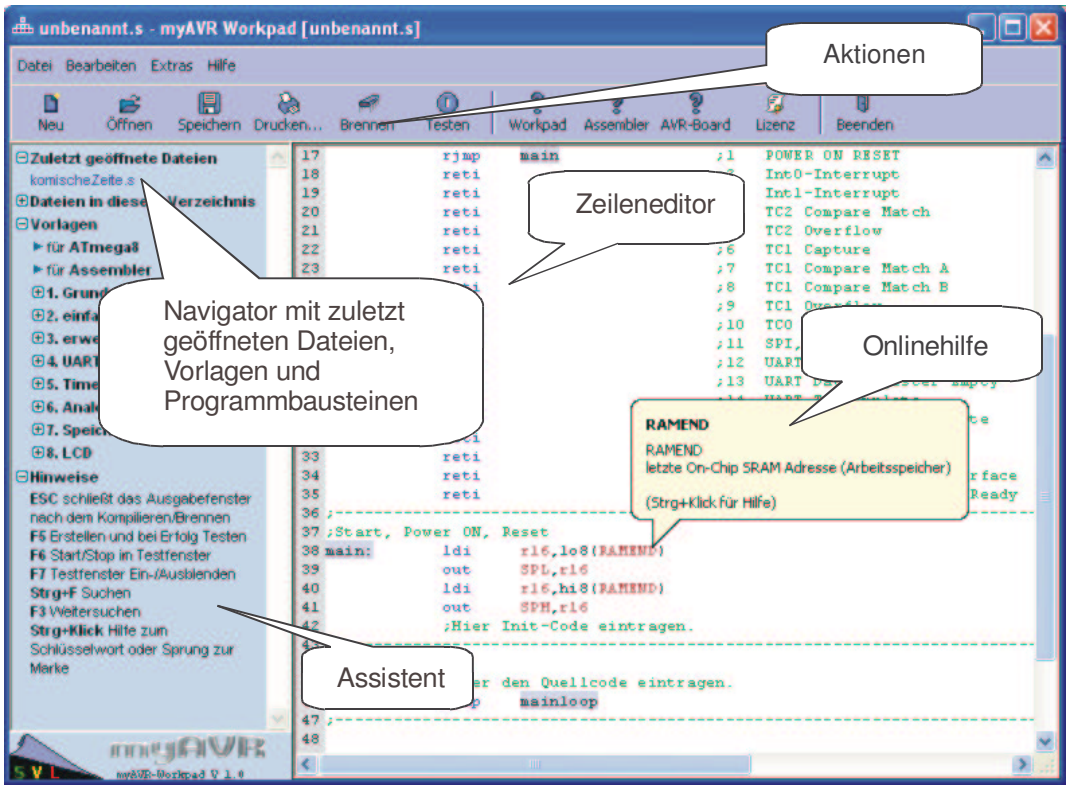


- Wenn vorhanden, ziehen Sie die Batterie bzw. das Netzteil und das Programmierkabel ab.
- Verbinden Sie die LEDs mit dem Prozessorport D entsprechend dem folgenden Schema.
  Nutzen Sie die Patchkabel!
- Prüfen Sie die Verbindungen und schließen Sie die Batterie/das Netzteil oder das Programmierkabel wieder an und nehmen Sie die Mikrocontrollerlösung in Betrieb.
- Es sind jetzt alle LED's an.
- Gratulation!
  Das ist Ihre erste Mikrocontrollerlösung mit dem myAVR Board.

Abbildung: myAVR Controlcenter

4.4 Entwicklungsumgebung myAVR Workpad

Diesen und den folgenden Abschnitt können Sie überspringen, wenn Sie nicht das myAVR Workpad als Entwicklungsumgebung verwenden.
Schauen wir uns als nächstes kurz in der Entwicklungsumgebung myAVR Workpad um. Das myAVR Workpad ist ein einfaches Entwicklungswerkzeug, mit dem man schnell und unkompliziert Mikrocontrollerlösungen programmieren kann.



Beim Compilieren, Linken und Brennen des Schnellstart-Beispiels öffnete sich ein Ausgabefenster und zeigte Protokollausgaben der Aktionen an. Wenn die Hardware ordnungsgemäß angeschlossen, von der Software erkannt und das Programm erfolgreich auf den Programmspeicher des Mikrocontrollers übertragen wurde, muss die letzte Ausschrift folgenden Inhalt haben:

```
Hinweis: Es wird die im Quelltext angegebene Einstellung des
Prozessors verwendet: ATmega8
assemblieren...........
linken...........
brennen...........

Found programmer: ...
```

Die Inbetriebnahme, Test und Datenkommunikation mit der Mikrocontrollerlösung erfolgen über das myAVR-Controlcenter. Dabei wird über die Schaltfläche „Start“ das Testboard mit der nötigen Betriebsspannung versorgt und der Controller gestartet.

Der Datenaustausch mit dem myAVR Board ist möglich, wenn das Null-Modemkabel (oder USB-Kabel) an Rechner und Testboard angeschlossen ist, sowie die Mikrocontrollerlösung dafür vorgesehen ist. Es können Texte und Bytes (vorzeichenlose ganzzahlige Werte bis 255) an das Board gesendet und Text empfangen werden. Die empfangenen Daten werden im Protokollfenster angezeigt.

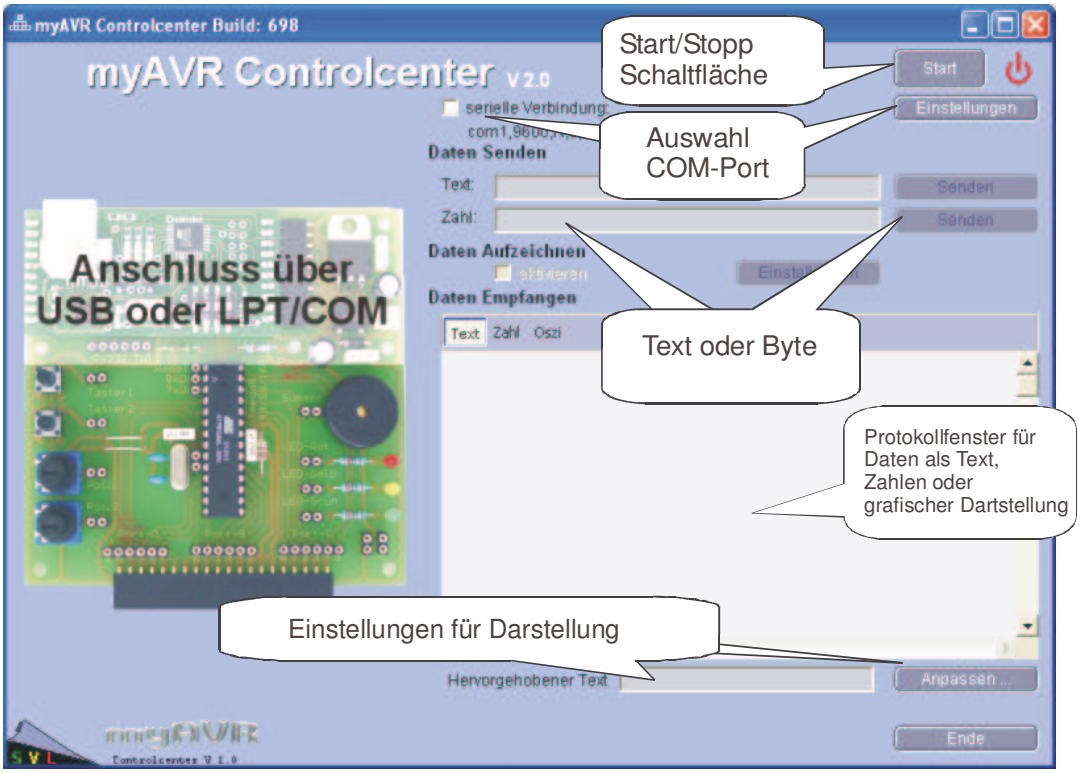


Abbildung: myAVR Controlcenter

Nutzen Sie die zahlreichen Hilfen und Vorlagen, die das myAVR Workpad dem Entwickler bietet!

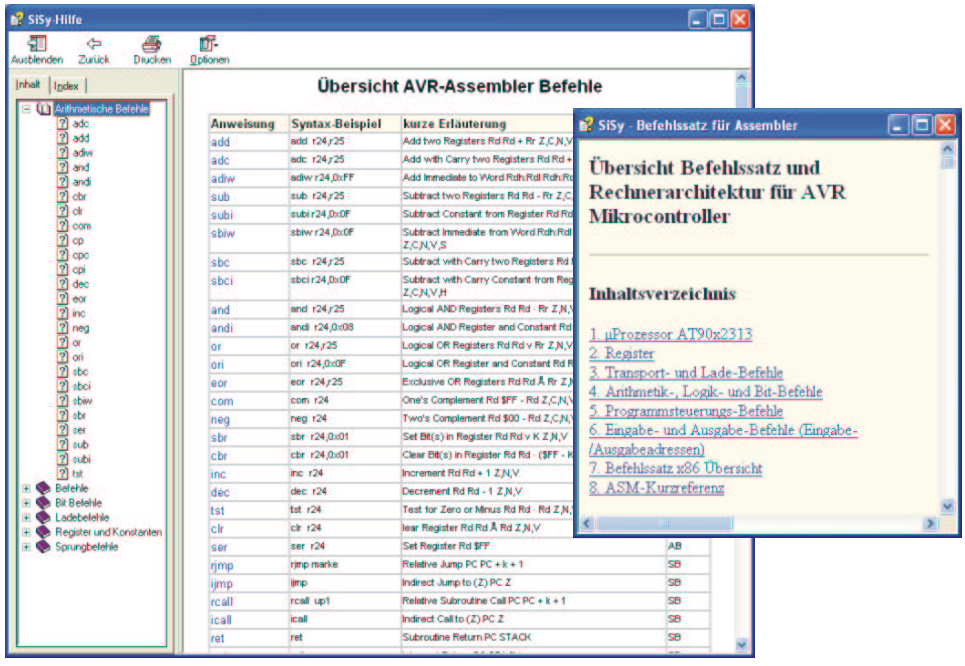


Abbildung: AVR-Assembler --> Dictionary und Hilfe

4.5 Schnellstart mit dem myAVR Workpad

Voraussetzungen:

Für die Bearbeitung der folgenden Aufgaben benötigen Sie die aufgeführte Software und Hardware.

Software:

- myAVR Workpad

Hardware:

- Ein bestücktes myAVR Board
- LPT-Verlängerungskabel oder USB-Kabel A-B
- 9 V Batterie/Netzteil bei Bedarf (z.B.: autonomer Einsatz)
- evtl. Nullmodemkabel für die COM Schnittstelle

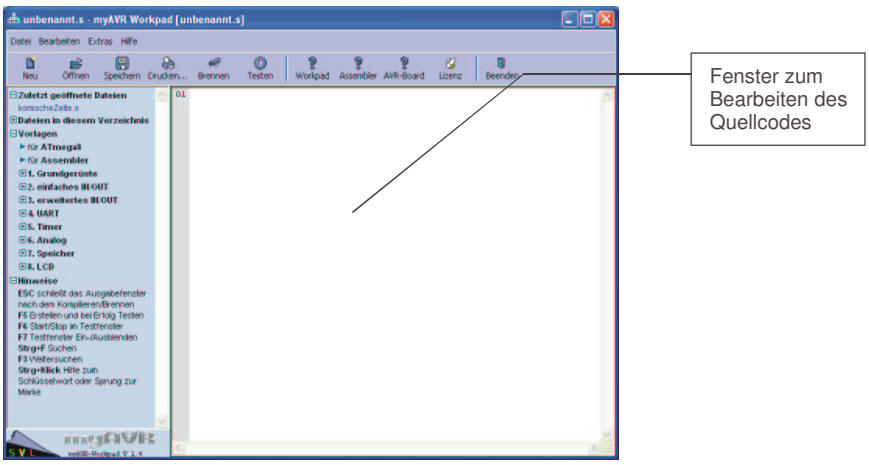
Zielstellung:

Der Schnelleinstieg zur Mikrocontroller-Programmierung soll Ihnen helfen, myAVR Workpad kennen zu lernen und erste Schritte in der hardwarenahen Programmierung mit diesem Programm zu gehen. Die Funktion des nachfolgenden Programms ist es, beim Start alle optischen Ausgabegeräte (rote, grüne und gelbe LED) auf dem myAVR Board zum leuchten zu bringen.

Ein neues Projekt anlegen

Starten Sie das myAVR Workpad und wählen Sie „Neue Datei*“.* Speichern Sie die Datei unter dem Namen „Alle_Lichter_an“.

SiSy - Windows Application mit SVL
Willkommen im myAVR Workpad SE
(Build 1370)
Dieses Workpad ist speziell für die einfache Handhabung und schnellen Einstieg in die AVR-Programmierung konzipiert. So können Sie ohne komplizierte Einstellungen vorzunehmen sofort starten.
Programm-Grundgerüst, Beispiele und Vorlagen unterstützen Sie beim Programmieren.
zuletzt geöffnete Dateien
geändert
komischeZeile.s
16.03.2006 13:45:57
Datei suchen
Neue Datei
mehr unter
www.myAVR.de
Schaltfläche „Neue Datei“ wählen



Assembler-Programm anlegen

Wählen Sie über den Assistenten unter Grundgerüste den Punkt *Grundgerüste, Hauptprogramm.* Laden Sie die Vorlage für das Programmgerüst oder erstellen Sie den entsprechenden Quellcode.

```
;-------------------------------------------------------------
;* Titel            :Programmvorlage für myAVR Board
;* Prozessor        :ATmega8 mit 3,6864 MHz
;* Schaltung        :...
;* Datum            :15.03.2004
;* Autor            :Dipl. Ing. Päd. Alexander Huwaldt
;-------------------------------------------------------------
.include     "avr.h"
;-------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:        rjmp    main        ; 1    POWER ON RESET
              reti                ; 2    Int0-Interrupt
              reti                ; 3    Int1-Interrupt
              reti                ; 4    TC2 Compare Match
              reti                ; 5    TC2 Overflow
              reti                ; 6    TC1 Capture
              reti                ; 7    TC1 Compare Match A
              reti                ; 8    TC1 Compare Match B
              reti                ; 9    TC1 Overflow
              reti                ; 10   TC0 Overflow
              reti                ; 11   SPI, STC Serial Transfer Complete
              reti                ; 12   UART Rx Complete
              reti                ; 13   UART Data Register Empty
              reti                ; 14   UART Tx Complete
              reti                ; 15   ADC Conversion Complete
              reti                ; 16   EEPROM Ready
              reti                ; 17   Analog Comparator
              reti                ; 18   TWI (I²C) Serial Interface
              reti                ; 19   Store Program Memory Ready
;-------------------------------------------------------------
; Start, Power ON, Reset
main:         ldi    r16 , lo8(RAMEND)
              out    SPL , r16                ; Init Stackpointer LO
              ldi    r16 , hi8(RAMEND)
              out    SPH , r16
              ; Hier den Init-Code eintragen
;-------------------------------------------------------------
mainloop:     wdr
              ; Hier den Quellcode eintragen
              rjmp mainloop
;-------------------------------------------------------------
```

Abbildung: Grundgerüst AVR-Assembler-Programm

Quellcode in Assembler erstellen

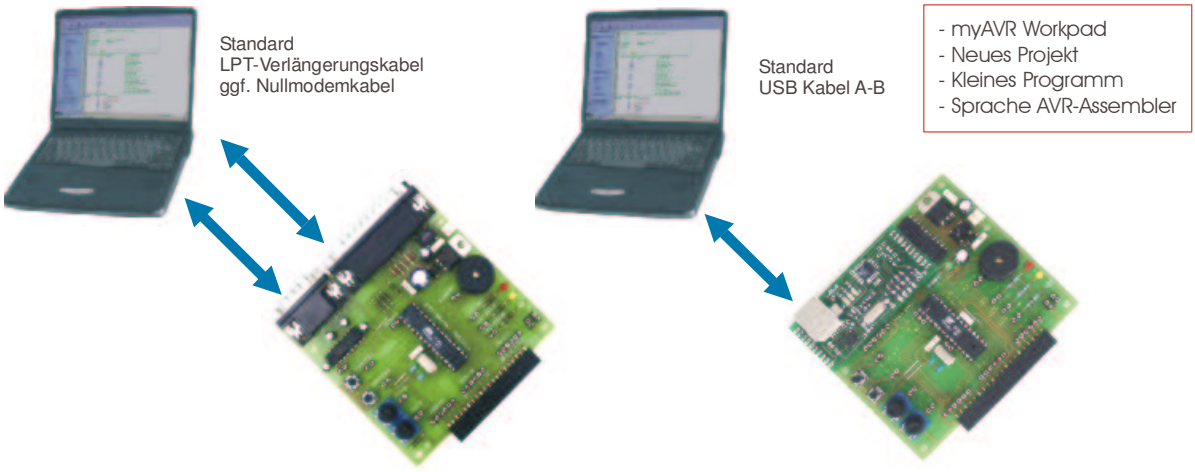
Die Ausgabegeräte LED sollen vom Prozessorport D gesteuert werden. Die Realisierung erfolgt über Bits im Register R16. Geben Sie folgenden Quellcode ein bzw. ergänzen Sie die Programmvorlage.

```
;+-----------------------------------------------------+
;| Titel          : Übung: „alle Lichter an"           |
;+-----------------------------------------------------+
;| Prozessor      : ATmega8  3,6864 MHz                |
;| Sprache        : Assembler                          |
;| Datum          : 28.02.2006                         |
;| Autor          : Dipl. Ing. Päd. Alexander Huwaldt  |
;+-----------------------------------------------------+
.include    "AVR.H"
;-------------------------------------------------------
;Reset and Interrupt vector ;VNr.  Beschreibung
   rjmp    main              ;1     POWER ON RESET
   reti                      ;2     Int0-Interrupt
   reti                      ;3     Int1-Interrupt
   reti                      ;4     TC2 Compare Match
```

```
   reti                    ;5     TC2 Overflow
   reti                    ;6     TC1 Capture
   reti                    ;7     TC1 Compare Match A
   reti                    ;8     TC1 Compare Match B
   reti                    ;9     TC1 Overflow
   reti                    ;10    TC0 Overflow
   reti                    ;11    SPI, STC Serial Transfer Complete
   reti                    ;12    UART Rx Complete
   reti                    ;13    UART Data Register Empty
   reti                    ;14    UART Tx Complete
   reti                    ;15    ADC Conversion Complete
   reti                    ;16    EEPROM Ready
   reti                    ;17    Analog Comparator
   reti                    ;18    TWI (IC) Serial Interface
   reti                    ;19    Store Program Memory Ready
;-----------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     r16,0b00011100      ; Bit 2-4 auf HIGH
            out     DDRD,r16            ; Port D.2-4 auf OUT
;-----------------------------------------------------------
mainloop:   wdr
            ldi     r16, 0b00011100     ;  Bit 2-4 auf HIGH
            out     PORTD,r16           ;  Port D.2-4 auf HIGH
            rjmp    mainloop
```

Hardware anschließen und brennen

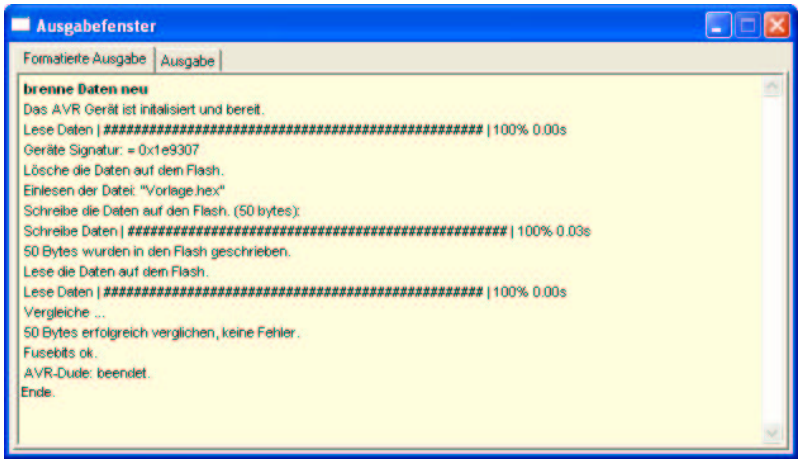
Das myAVR Board verfügt über eine ISP (In System Programming) Schnittstelle. Der Prozessor muss also nicht für die Programmierung aus dem System entfernt werden, um ihn in einem gesonderten Programmiergerät zu brennen, sondern kann im myAVR Board direkt programmiert werden. Dazu schließen Sie das Programmierkabel an den LPT- oder USB-Port Ihres Rechners.



myAVR Board 1 LPT **myAVR Board 2 USB**

Compilieren, Linken und Brennen

Der eingegebene Quellcode muss nun in Maschinencode für den AVR-Prozessor übersetzt werden. Wählen Sie dazu die Schaltfläche „Brennen“. Bei fehlerfreier Übersetzung liegt das Programm als „Alle_Lichter_An.hex“ vor und kann auf den FLASH-Programmspeicher des Prozessors gebrannt werden. Bei erfolgreichem Brennvorgang erhalten Sie im Ausgabefenster eine entsprechende Meldung. Der Umfang der Informationen variiert bei Verwendung des LPT- bzw. USB-Boards.


Ausgabefenster
Formatierte Ausgabe
Ausgabe
brenne Daten neu
Das AVR Gerät ist initialisiert und bereit.
Lese Daten | ########## | 100% 0.00s
Geräte Signatur: = 0x1e9307
Lösche die Daten auf dem Flash.
Einlesen der Datei: "Vorlage.hex"
Schreibe die Daten auf den Flash. (50 bytes):
Schreibe Daten | ########## | 100% 0.03s
50 Bytes wurden in den Flash geschrieben.
Lese die Daten auf dem Flash.
Lese Daten | ########## | 100% 0.00s
Vergleiche ...
50 Bytes erfolgreich verglichen, keine Fehler.
Fusebits ok.
AVR-Dude: beendet.
Ende.

Mikrocontrollerlösung testen

Um das Programm zu testen ist es nötig, den Port D mit den Ausgabegeräten LED zu verbinden.

- Ziehen Sie ggf. die Batterie/das Netzteil und das Programmierkabel ab.
- Verbinden Sie die LEDs mit dem Prozessorport D entsprechend dem folgenden Schema.
  Nutzen Sie die Patchkabel!
- Prüfen Sie die Verbindungen und schließen Sie die Batterie/das Netzteil oder das Programmierkabel wieder an und nehmen die Mikrocontrollerlösung in Betrieb.
- Es sind jetzt alle LED's an.
- Gratulation!
  Das ist Ihre erste Mikrocontrollerlösung mit dem myAVR Board.

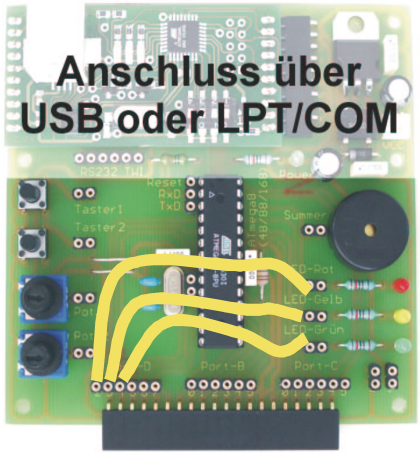


Abbildung: myAVR Controlcenter

▪▪▪

5.4 Das seltsame „Hallo Welt“ eines Mikrocontrollers

Der Anfang aller Programmierung ist das berühmte „Hallo Welt“ auf dem Bildschirm. Die Zielstellung dieses Programms ist es den minimalen Grundaufbau eines Programms und den Erstellungszyklus: Programmieren, Kompilieren, Linken und Starten/Testen des Programms zu erlernen. Dabei soll natürlich etwas zu sehen sein. Also beginnt man mit einer einfachen Textausgabe auf dem Bildschirm. Mikrocontrollerprogramme sind aber auf dem PC nicht ausführbar und ein Bildschirm ist an vielen Mikrocontrollerlösungen gar nicht vorgesehen. Als Ausgabegerät hat man oft nur eine LED zur Verfügung. Das „Hallo Welt“ eines Mikrocontrollers ist in unserem Fall nur eine LED die eingeschaltet wird oder blinkt. Dies soll die erste zu lösende Aufgabe sein.

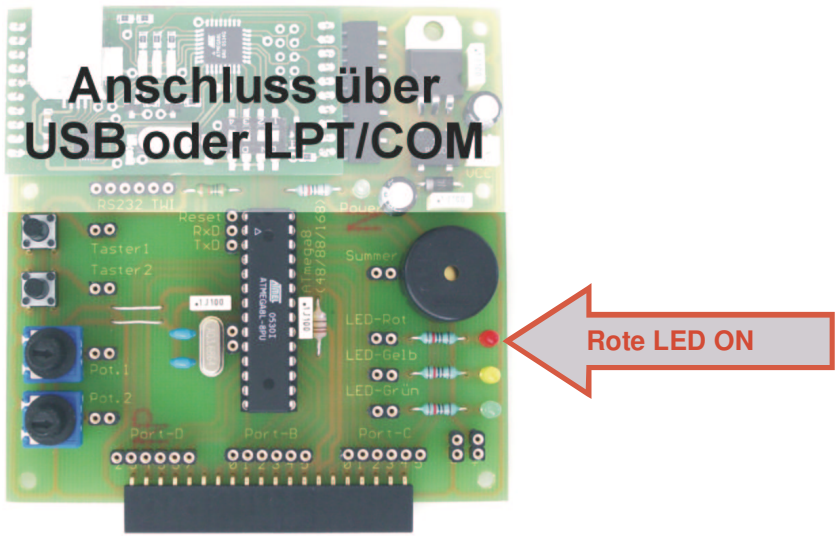


Abbildung: Das seltsame „Hallo Welt“ der µController

Die vorbereitenden Arbeiten für jede Übung sind das Anschließen der Hardware und Starten der Entwicklungsumgebung.

1. Hardware anschließen
2. SiSy AVR oder myAVRWorkpad starten
3. Neues Projekt/Datei anlegen
4. Sprache AVR-Assembler wählen

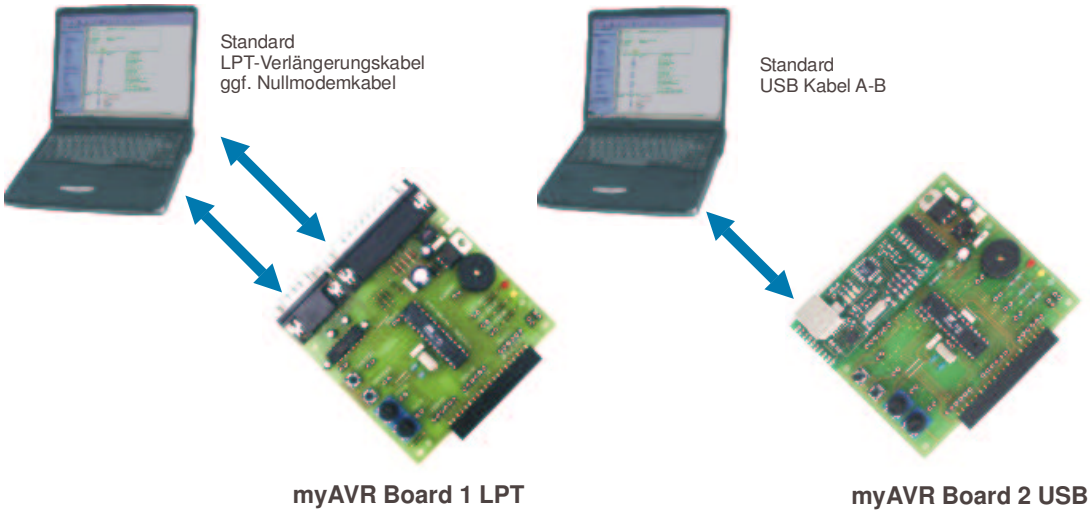


Abbildung: Hardware anschließen (LPT- und USB-Variante)

Technische Aspekte für das seltsame „Hallo Welt“

Die Aufgabe ist dem Wesen nach die Ausgabe einer Information. Das Ausgabegerät ist die rote LED. Der im myAVR Board verwendete Controller verfügt über drei digitale 8 Bit parallele Ein- und Ausgabeports. Jedes Bit repräsentiert eine I/O-Line (Pin) und ist einzeln als Eingang oder Ausgang konfigurierbar.

- **`PORTB  0x18  Port B.0 bis Port B.7`**
- **`PORTC  0x15  Port C.0 bis Port C.7`**
- **`PORTD  0x12  Port D.0 bis Port D.7`**

Es ist zu beachten, dass die einzelnen Ports des Controllers für spezielle Aufgaben vorgesehen sind und dass die meisten I/O-Lines (Pins) doppelt belegt sind. Diese Doppelbelegung sorgt zwar dafür, dass die Baugröße des Controllers gering ist aber führt auch dazu, dass nicht alle Bits der Ports für die Anwendung uneingeschränkt zur Verfügung stehen.
Wichtige Besonderheiten:

- **`Port B:`** **`Port B.6`** und **`Port B.7`** sind für den externen Quarz vorgesehen.
  `Port B.5`, **`Port B.4`** und **`Port B.3`** sind gleichzeitig die SPI-Schnittstelle zur Programmierung.
- **`Port C:`** **`Port C.7`** ist nicht als I/O-Line verfügbar.
  `Port C.6` ist mit `RESET` belegt (nie umprogrammieren!).
  `Port C.0` bis **`Port C.5`** sind gleichzeitig die Eingänge für den A/D-Wandler.
- **`Port D:`** **`Port D.0`** und **`Port D.1`** sind die Leitungen für den UART Kommunikationsbaustein.

Daraus ergeben sich wichtige Besonderheiten für die konkrete Nutzung der Ports auf dem myAVR Board.

- je Port sind 6 Lines frei nutzbar
  - Port D.2 bis Port D.7
  - Port B.0 bis Port B.5
  - Beachte für Port B.3 bis Port B.5 bei LPT-Version unter Umständen das Programmierkabel abziehen!
  - Port C.0 bis Port C.5, bevorzugt als ADC

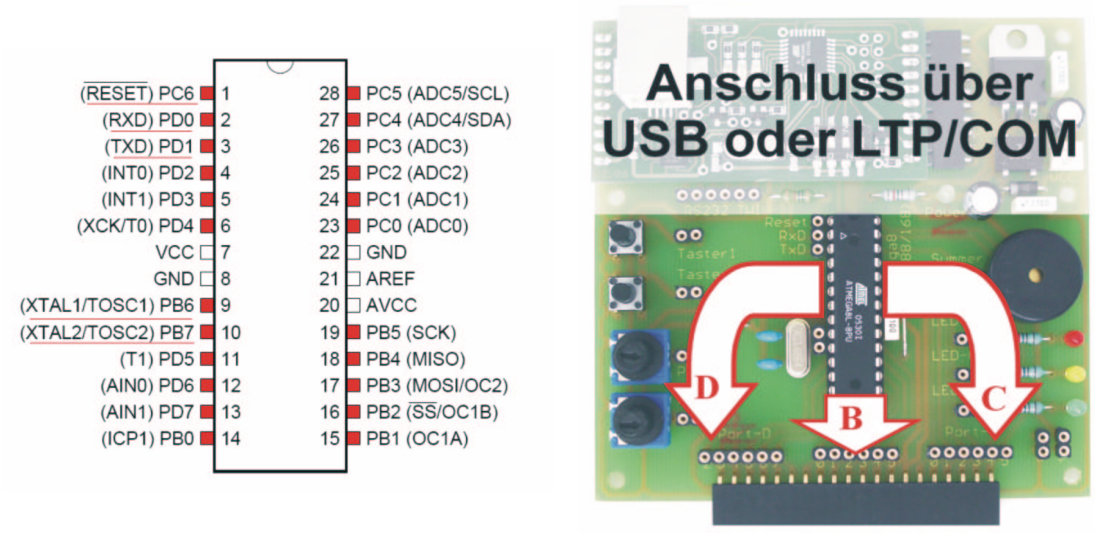


Abbildung: Digitale I/O-Ports auf dem myAVR Board

Auf dem myAVR Board ist die RS232 Schnittstelle für das Verbinden mit dem COM-Port eines PC bereits fest installiert. Dafür ist ein 9-poliger SUB-D Stecker und ein Treiberbaustein (MAX232) implementiert. Die UART Leitungen zum Empfang und für das Senden von Daten (RxD und TxD) sind fest mit dem Treiberbaustein verbunden. Damit stehen die Leitungen für Port D.0 und Port D.1 für die freie Verwendung nicht zur Verfügung. Die SPI Schnittstelle ist ebenfalls vollständig auf dem Board realisiert und mit dem 25-poligen SUB-D Stecker für den Anschluss an den Parallelport eines PC verbunden. Dadurch ist ein zusätzlicher Programmer nicht mehr erforderlich.
Zu Beachten ist, dass unter Umständen die SPI-Leitungen Port B.3 bis Port B.5 nur dann benutzt werden können, wenn das LPT-Kabel abgezogen ist. Die Doppelbelegung des Port C.6 mit der `RESET`-Leitung führt dazu, dass diese ebenfalls nicht zur Verfügung steht. Es ist zwar möglich die `RESET`-Leitung abzuschalten und damit Port C.6 nutzbar zu machen, aber von dem Augenblick an lässt sich der Controller nicht mehr umprogrammieren, denn dafür benötigt er ein `RESET`-Signal. Der A/D-Wandler an Port C benötigt eine eigenständige Versorgungsspannung. Dies muss bei der Nutzung des Port C beachtet werden.

Jede einzelne Leitung der digitalen Ports ist geeignet, eine bestimmte Last anzusteuern. Eine besondere Eigenschaft von Mikrocontrollern ist, dass deren Ausgänge eine verhältnismäßig „große“ Leistung abgeben können. Je nach Controllertyp können das je Pin 3 bis 20 mA bei 5 V HI-Pegel sein. Mit 3 mA kann man eine kleine LED oder einen hochohmigen Schallwandler problemlos ansteuern. Der Widerstand des so angesteuerten Gerätes sollte nur nicht 250 Ω unterschreiten. Bei einer LED ist es notwendig, den Strom durch einen Vorschaltwiderstand von 1,2 kΩ zu begrenzen und schon kann man die LED mit dem gewünschten Port verbinden und ansteuern. Die Programmierung der Ports erfolgt bei allen AVR-Mikrocontrollern über jeweils drei I/O-Register. Jedem Port ist ein Steuerregister, ein Ausgaberegister und ein Eingaberegister zugeordnet. Das Steuerregister hat die Bezeichnung `DDRx`, Data Direction Register Port B/C/D. Es legt also die Richtung des Datenflusses fest. Der Programmier ist über dieses Register in der Lage, jeden einzelnem Pin der digitalen Ports getrennt als Eingang oder als Ausgang zu konfigurieren. Die Ausgabe von Daten oder das Ansteuern von Geräten erfolgt über das jeweilige Register `Port x`, die Eingaben über das Register `Pin x`.

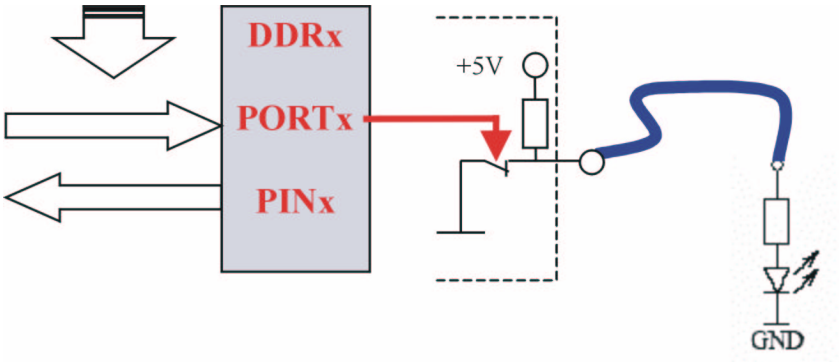


Abbildung: Schema einer digitalen I/O-Line mit angeschlossener LED

Port	B	C	D
Steuerregister	DDR B , 0x17	DDR C, 0x14	DDR D, 0x11
Ausgaberegister	Port B, 0x18	Port C, 0x15	Port D, 0x12
Eingaberegister	Pin B, 0x16	Pin C, 0x13	Pin B, 0x10

Für die Ausgabe über einen der digitalen Ports sind folgende Schritte notwendig:
1. Port/Line auf Eingang oder Ausgang konfigurieren
2. Ausgabe

Die Konfiguration des Ports erfolgt in der Regel während der Initialisierungssequenz über das Register DDRx. Dabei steht jedes Bit in diesem Steuerregister für die Daten-Richtung der einzelnen I/O-Lines. Eine logische 0 bewirkt, dass der entsprechende Prozessor-Pin als „Eingang“ konfiguriert ist. Die logische 1 in einem Bit bewirkt, dass der entsprechende Prozessor-Pin als „Ausgang“ konfiguriert ist. Dafür stehen grundsätzlich zwei Methoden zur Verfügung:

1. Konfiguration per **out** Befehl
   a. allgemeines Register mit kompletter I/O Konfiguration laden (z.B.: `ldi`)
   b. Ausgabe der Portkonfiguration mittels `out`

```
;Beispiel: Port B.0 = Ausgang, Port B.1 bis 7 Eingang
                    ldi   r16, 0b00000001
                    out   DDRB , r16
```

2. Konfiguration per **sbi** Befehl

```
;Beispiel: Port B.0 = Ausgang, Port B.1 bis 7 unverändert
      sbi   DDRB,0
```

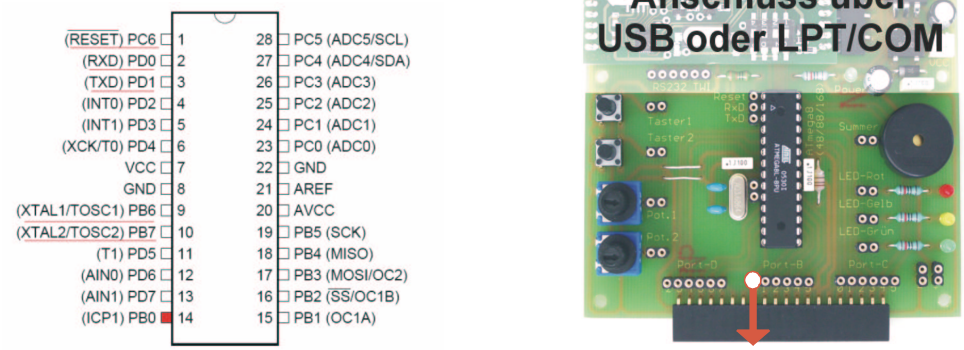


Abbildung: Konfiguration des Port B.0 als Ausgang

Die vorgestellten Befehle haben folgende Wirkung:

- `ldi Register`, Konstante
  - lade eine Konstante in ein allgemeines Register
  - wirkt nur auf die Register r16 bis r31

- `out Port/IO-Register`, Register
  - gebe allgemeines Register an Portx aus
  - Ziel immer ein I/O Register oder Port
  - Quelle immer ein allgemeines Register r0 bis r31

- `sbi Port/IO-Register`, Bit
  - setze Bit in I/O-Register oder Port
  - Ziel ist immer ein I/O-Register oder Port
  - Parameter: die Bit-Stelle 0 bis 7

Der wesentliche Unterschied zwischen beiden Methoden besteht darin, dass bei der Verwendung des Befehls `out` immer der gesamte Port konfiguriert wird. Dabei kann es auch möglich sein, dass versehentlich eine andere Konfiguration über schrieben wird.
Beim Befehl `sbi` wird gezielt nur ein Bit gesetzt. Eventuell vorher gesetzte Konfigurationen anderer I/O-Lines bleiben unberührt. Dieser Befehl ist in der Regel vorzuziehen.

Die Ausgabe selbst kann wiederum mit den oben besprochenen Befehlen erfolgen, da programmiertechnisch der reale Ausgabeport genauso gehandhabt wird wie die Steuerregister. Das Ausgaberegister ist entsprechend Port x.

1. Ausgabe per **`out`** Befehl
   a. allgemeines Register: Ausgabedaten laden (z.B.: `ldi`)
   b. Ausgabe an den Port mittels `out` senden.
   Beachten Sie: Nur die als Ausgang konfigurierten Bits werden „durchgelassen“.

```
; Beispiel: Port B.0 = High
          ldi   r16, 0b00000001
          out   PORTB , r16

; Beispiel: Port B.0 = Low
          ldi   r16, 0b00000000
          out   PORTB , r16
```

2. Ausgabe per **`sbi`** Befehl

```
; Beispiel: Port B.0 = High
          sbi   PORTB , 0

; Beispiel: Port B.0 = Low
          cbi   PORTB , 0
```

Hinweis: Die Angabe der Bit auf Lines, die nicht auf Ausgabe geschaltet sind, ist ohne Effekt.

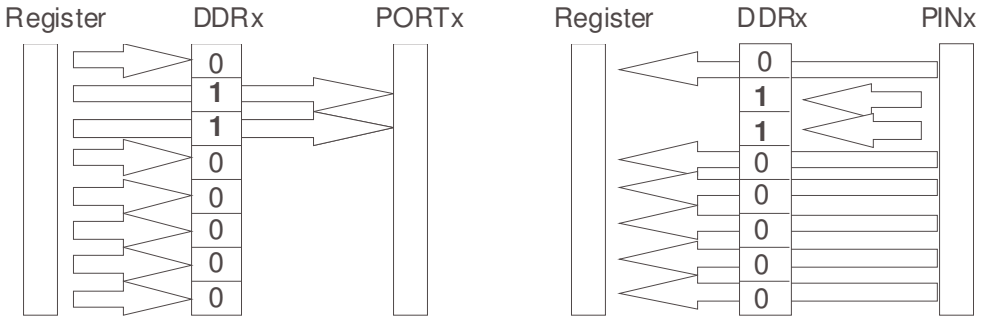
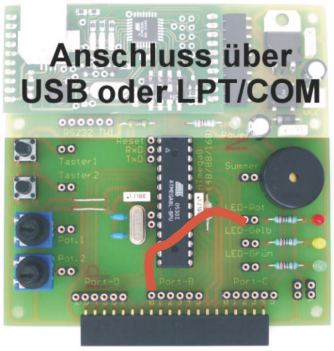


Abbildung: schematische Wirkung des Steuerregisters DDRx bei out und in

Schaltung:

Um die rote LED einzuschalten, muss eine Leitung des gewünschten Port x mit der roten LED verbunden werden. Nutzen Sie eines der mitgelieferten Patch-Kabel. Verbinden Sie Port B.0 mit der roten LED.



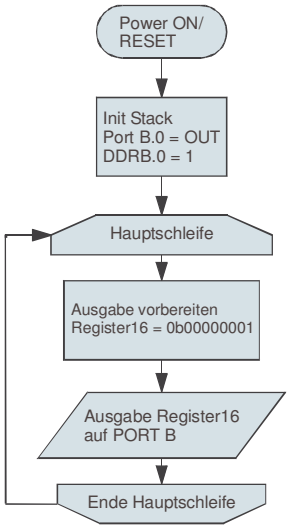


Abbildung: Schaltung und Programmablauf für das Beispiel „Hallo Welt“

Lösungsansatz:

Eines der allgemeinen Register (z.B.: r16) wird als temporäre Variable verwendet. Das Bit 0 vom Port B wird über das Register DDR B auf „Ausgabe“ geschaltet (`sbi DDRB,0`). Die Ausgabe erfolgt in zwei Schritten:

- laden des Ausgabewertes in ein allgemeines Register (`ldi r16,0b00000001`) und
- die eigentliche Ausgabe (`out PORTB,r16`).

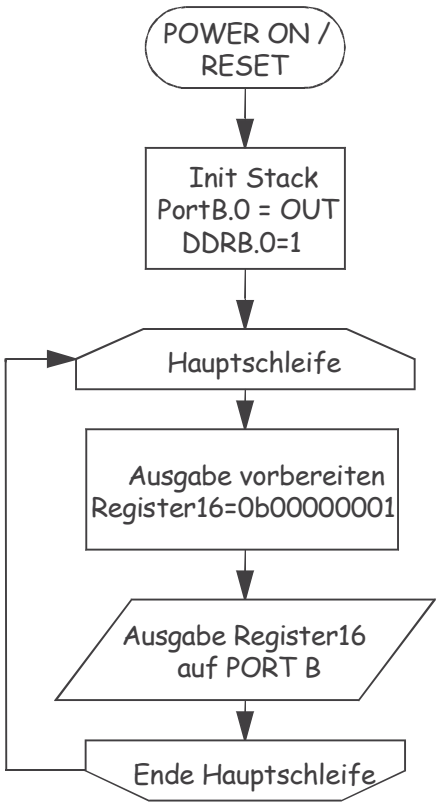
```
;-----------------------------------------------------------------------------
;* Titel              :Beispiel "Hallo Welt" für myAVR Board
;* Prozessor          :ATmega8 mit 3,6864 MHz
;* Datum, Autor       :15.03.2004, Dipl. Ing. Päd. Alexander Huwaldt
;-----------------------------------------------------------------------------
.include       "avr.h"
;-----------------------------------------------------------------------------
; Reset and Interruptvectoren        ; VNr.  Beschreibung
begin:         rjmp    main        ; 1     POWER ON RESET
               reti                ; 2     Int0-Interrupt
               reti                ; 3     Int1-Interrupt
               reti                ; 4     TC2 Compare Match
               reti                ; 5     TC2 Overflow
               reti                ; 6     TC1 Capture
               reti                ; 7     TC1 Compare Match A
               reti                ; 8     TC1 Compare Match B
               reti                ; 9     TC1 Overflow
               reti                ; 10    TC0 Overflow
               reti                ; 11    SPI, STC Serial Transfer Complete
               reti                ; 12    UART Rx Complete
               reti                ; 13    UART Data Register Empty
               reti                ; 14    UART Tx Complete
               reti                ; 15    ADC Conversion Complete
               reti                ; 16    EEPROM Ready
               reti                ; 17    Analog Comparator
               reti                ; 18    TWI (I²C) Serial Interface
               reti                ; 19    Store Program Memory Ready
;-----------------------------------------------------------------------------
; Start, Power ON, Reset
main:          ldi     r16 , lo8(RAMEND)
               out     SPL , r16
               ldi     r16 , hi8(RAMEND)
               out     SPH , r16           ; init Stack
               sbi     DDRB ,0
;-----------------------------------------------------------------------------
mainloop:      ldi     r16 , 0b00000001    ; LED ON
               out     PORTB , r16
               rjmp    mainloop
;-----------------------------------------------------------------------------
```

Realisierung:
Geben Sie den Quellcode ein. Benutzen Sie das Grundgerüst für ein AVR-Assemblerprogramm und ergänzen Sie dieses. Kompilieren, Linken und Brennen Sie das Programm. Führen Sie die Anwendung aus.
Fertigen Sie nach erfolgreicher Übung bitte ein Protokoll an. Sie werden den Wert der Protokolle bald erkennen.

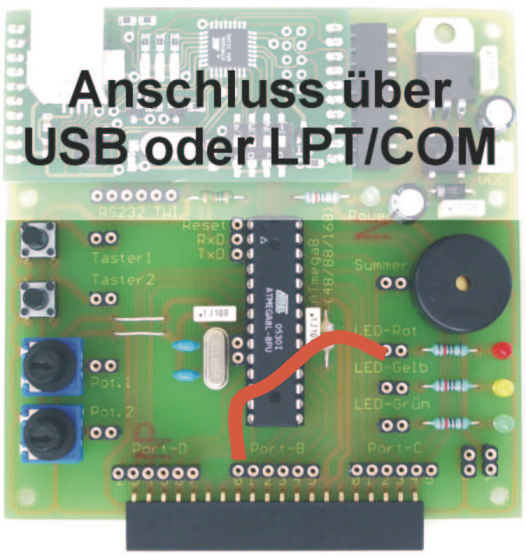
Protokoll *Hallo Welt* **AVR-Assembler**

Übungsaufgabe: *rote LED einschalten*

Lösungsentwurf:



Schaltung:



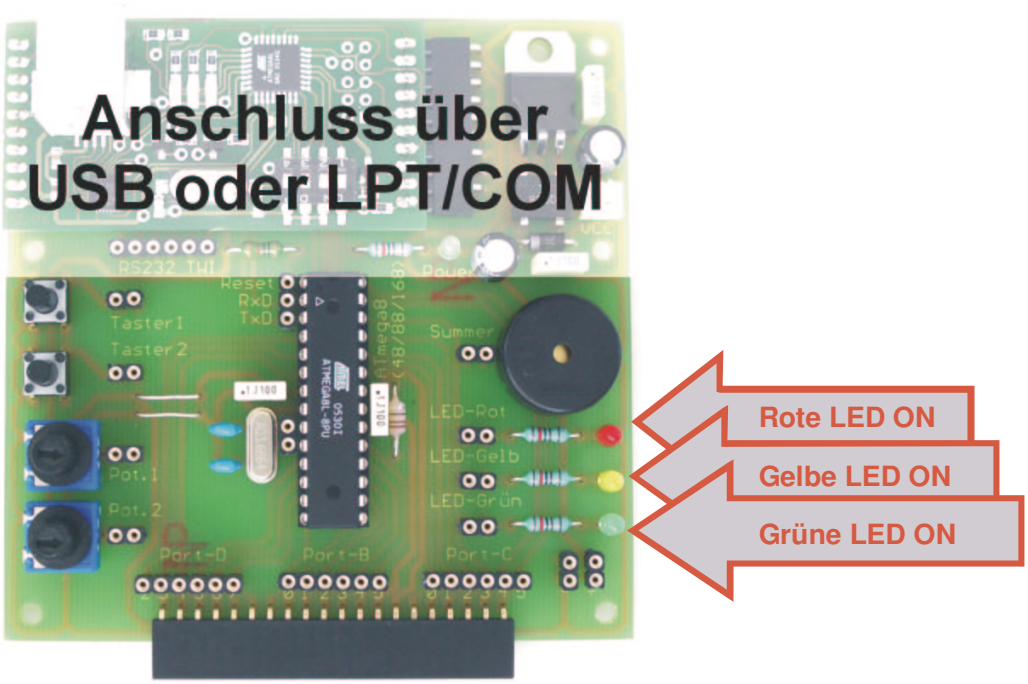
Notitzen:
- ...

Quellcode:

```
...
main:       ldi     r16 , lo8(RAMEND )      ; Konstante RAMEND, Adresse Ende SRAM
            out     SPL , r16               ; Initialize Stackpointer mit RAMEND LOW
            ldi     r16 , hi8(RAMEND )      ; Konstante RAMEND, Adresse Ende SRAM
            out     SPH , r16               ; Initialize Stackpointer mit RAMEND HIGH
            sbi     DDRB , 0                ; konfiguriere Port B.0  = 1 auf Ausgang
;-----------------------------------------------------------------------
mainloop:   ldi     r16 , 0b00000001        ; Ausgabedaten in allgemeines Register
            out     PORTB , r16             ; Ausgabe der Daten
            rjmp    mainloop
```

Übung: „alle Lichter an“

Lösen Sie die folgende Aufgabe selbständig. Es sollen die rote, gelbe und grüne LED eingeschaltet werden. Legen Sie dazu ein neues „kleines Programm“ in Ihrem Projekt an. Gehen Sie wie oben beschrieben vor und protokollieren Sie die Übung.



Sachwortverzeichnis

A

Algorithmus 70
Analog/Digital-Wandler 137, 177
Analoge Signale 21
Applikationen 14
Assemblerbefehlszeile 48
AVR-Controlcenter 34

B

BASCOM 161
BASCOM Befehlssatz 165
BAUD-Rate 115
Befehlsgruppen 47
Brennen 32, 166

C

CISC-Prozessor 20
Comparator 18, 131, 158, 176
Compilieren 32, 166
C-Programm 155

D

Datenübertragung 113
Digitale Signale 21
Doppelbelegung 50

E

EEPROM 20, 127
Eingabe analog .. Siehe Comparator & Analog/Digital-Wandler
Eingabe digital 58
Eingaben in BASCOM 170
Einsprungpunkt 44
Entprellen 96

F

FLASH 122
Frequenzen 100

H

Hardware 12
Hauptschleife (Betriebssystem) 46

I

In System Programming (ISP) 26
Initialisierungssequenz 45
Interrupt 19, 44, 88, 92
Interruptserviceroutine 46, 90
Interrupt-Steuerung 86, 88
Interruptvektor 44, 88, 159

K

Kommunikation mit einem PC 114
Konstanten im Programmspeicher 121

L

Linken 32, 38

M

myAVR-Controlcenter 29

N

Neues Projekt anlegen 30, 36

P

Port 41
Prellen 96
Programmkopf 44
Programmvariable 119, 156, 166
Pull-up 58
Pulsweitenmodulation 145

R

Register 19, 41, 119
RISC-Prozessor 20

S

Software 15
Speicher 19, 43
Sprunganweisung 70
SRAM 124

T

Timer 105
Ton ..... Siehe Frequenz

U

UART 113, 157, 173
Unterprogramm 46
Unterprogramme 79, 165, 171
Unterprogrammtechnik 79

V

Variablennamen für Register 120
von-Neumann-Architektur 7

W

Watchdog-Timer 151

Z

Zahlenformate 40
Zähler ..... *Siehe* Timer